A performance analysis of a reaction diffusion simulation on SpatialOS

Joshua Hale

1402439

Computer Games Technology

CMP403

2018

School of Design and Informatics

Abertay University®

0. Contents

0.1 Figures

0.2 Tables

0.3 Listings

0.4 Abbreviations

CLI – Command Line Interface, FPS – Frames per Second, GPGPU – General Purpose Graphics Processing Unit, GPU – Graphics Processing Unit, KPI – Key Performance Indicator, MIMD – Multiple Instruction Multiple Data, SIMD – Same Instruction Multiple Data, SDK – Software Development Kit

1. Abstract

The complexity of natural, social and economic systems makes simulating them a computationally expensive task. Cloud computing allows the processing of a single simulation to be distributed across multiple machines. SpatialOS is a cloud computing platform which is primarily being developed for use in video games, however its potential for hosting scientifically accurate simulations is worth investigating. This paper sought to answer the question 'What are the performance bottlenecks of a computationally complex distributed simulation deployed at scale on SpatialOS?'. The aim was to develop a simulation which could be used to analyse the performance of SpatialOS and test the hypothesis that consistent performance would be achievable by deploying additional workers as the scale is increased, in an amount proportional to the complexity of the simulation model. The simulation was a reaction diffusion model which could be deployed at various scales across multiple 'worker' machines. The simulation's performance across these scales was analysed using real-time metrics to understand where bottlenecks may be. In response, optimisations were implemented so that the cause of the performance bottlenecks could be confirmed. It was found that the communication latency between workers added an additional layer to the complexity of the simulation which disproved the hypothesis that additional workers would maintain performance. Reducing cross-worker communication results in significant improvements in performance. These optimisations could be application specific or more broadly applicable, but reducing the amount of communication occurring should be a developer's primary concern. The aims of the study were met and the research question answered, although the results will always be specific to the type simulation being constructed as each has a different system characteristic as its performance limiting factor. It is proposed that further communication reduction techniques should be investigated, or the simulation structures should be modified so that they are inherently less dependent on frequent communication.

2. Introduction

There's many reasons why researchers create simulations to model natural systems. Some processes would be too disruptive to experiment on in-situ, for example traffic or network behaviours. Other processes are unethical to experiment with, such as the spread of disease. There's even processes that are just too complex to perform experiments on as they occur in nature. For some biological scenarios, the conditions to run physical tests are prohibitively difficult to maintain. For these reasons and more, researchers have looked at the rapidly growing power of computer simulations to understand more about the physical world.

SpatialOS is a new cloud based framework for running simulations dynamically on virtual machines. It provides a complete back-end allowing the developer to focus on the actual simulation model without having to spend a significant amount of time creating a robust networking solution. Issues such as workload distribution, network failure handling and resource management are all handled natively by the SpatialOS platform. In addition to this, a plugin is provided by Improbable (the developers of SpatialOS) for use with the Unity game engine, allowing for rapid prototyping of a cloud based simulation using this game engine.

However, SpatialOS is a relatively new technology and is still in its infancy in terms of implementation and optimisation. Few commercial products using the technology exist, many that do exist are still in the alpha or beta stage of development. Therefore, little testing has been carried out into the performance limitations of the platform.

Whilst gaming technologies can provide an excellent development environment for the construction of biological simulations there are distinct differences in the requirements of the technology from the perspective of scientific research and game development. Game development is almost entirely focused around creating a

cohesive player experience, often at the cost of high fidelity under the surface, whereas for scientific research accuracy is key.

It is therefore important to quantitatively analyse the performance of gaming technologies such as SpatialOS to ascertain their usefulness to the field of scientific research before funding and time are poured into development in earnest. By identifying key performance bottlenecks early on, considerations can be made in the design of a high resolution biological model which can then exploit the advantages that using a gaming technology can provide, without falling victim to the shortfalls which are considered acceptable to game developers.

This research seeks to answer the question:

> What are the performance bottlenecks of a computationally complex distributed simulation deployed at scale on SpatialOS?

The aim is to develop a simulation which can be used to analyse the performance of SpatialOS, identify bottlenecks and provide a basis for iterative optimisations in response to identified bottlenecks.

The key objectives are to create a cloud based simulation using the SpatialOS platform. The simulation will be based on entities which are each responsible for their own functionality. The functionality itself is of limited importance as the model will be used for performance testing rather than research into any specific simulation area. The model will be iteratively scaled and optimised to measure and improve the performance of the simulation.

The hypothesis proposed is that consistent performance will be achievable by deploying additional workers as the scale is increased, in an amount proportional to the complexity of the simulation model.

There are some boundaries that needed to be in place to ensure that the project met the aim within the given timeframe. Firstly, a focus was not placed on implementing

data capture from the simulation calculations themselves. As stated in the objectives, the results of the scientific functionality are not the purpose of the implementation. Instead a focus will be placed on developing the simulation so that its performance can be measured when it is deployed at scale. Additionally, implementing real-time interaction into the simulation is superfluous to the aims of the project. Whilst the advantage of using a platform such as SpatialOS alongside Unity is that interactivity can be provided through client applications, the actual functionality of real-time interactivity does not contribute to an understanding of the performance profile of the simulation running on a distributed system.

The following section will outline the current techniques used to perform complex simulations, along with their advantages and disadvantages. From there, an explanation of the development process of the experimental application is described. An in-depth description of the structure of a SpatialOS project is given to help shed light on the complexity of a distributed simulation. The aggregated results are summarized in the section following that. The raw data can be viewed in the appendix, but due to the volume and form of data collected, the data has been presented in the results in a manner that allows comparisons to be drawn between experiments. These results are then discussed in terms of what they reveal about the performance profile of this simulation when run on SpatialOS. Finally, suggestions for further research are presented based on the findings of this investigation.

3. Literature Review

This section aims to provide an understanding of the current state of large-scale simulations and the methods used to implement them. Through an understanding of why different systems are used for running simulations, the uses of emerging tech can be more clearly elucidated. An approach can then be created which tests a given system for its performance characteristics.

3.1 Supercomputers

Due to the complexity of the phenomenon that are desirable to simulate, as well as the coupled and interconnected nature of the physical world in general, massive computational power is required to run simulations at useful scale and resolution. Initially this computational power was achieved using supercomputers, machines comprising a high number of processors to perform multiple calculations in parallel (Phillips *et al.*, 2002). Modern supercomputers make use of a multiple instruction multiple data (MIMD) architecture. This allows for processors to be running different processes on a collection of data in parallel. This makes supercomputers perfect for a simulation which models multiple types of behaviours concurrently, however the cost and size of them mean they are only accessible to certain institutions. Even with this limitation, examples of supercomputers being used to run large scale simulations exist outside of research. *EVE Online* (CCP, 2013) is a massive space simulation where all logged in players exist in the same simulated universe, known as a shard. By using a supercomputer server cluster, the simulation can handle over 30,000 concurrent players in one shard (Stefanescu, 2006). Being able to sustain such a large-scale simulation allows for otherwise unthinkable gameplay mechanics, the only way this could have been achieved during EVE's lifetime (2003 onwards) was through the use of supercomputer-like technology (Drain, 2008).

3.2 GPU

The next stage in finding computational power came from using general purpose graphics processing units (GPGPU) to run parallel simulations. As graphic processing units (GPU) became more powerful to allow for better rendering of 3d graphics, their potential use as a general purpose computing powerhouse became apparent. Unlike modern supercomputers, GPGPUs use a same instruction multiple data (SIMD) architecture. This means that to fully utilise their power, programs that run in parallel should be executing the same sequence of commands at the same time using different data (Dematté and Prandi, 2010). If a simulation can be constructed in such a way that this condition is met, the parallel nature of GPGPUs can be harnessed. The specific characteristics of a simulation that is well suited to GPGPU implementation are that its processes are highly parallel, that it has high degree of arithmetic intensity (as GPU's have a large capacity for arithmetic calculations) and that the concurrent processes operate on the same data types.

The relatively affordable and easily accessible technology needed to create these machines meant that designing models in such a way that they are highly parallel became a priority and frameworks such as FLAME GPU have been created which help researchers construct simulations in this way (Richmond *et al.,* 2010). Reaction-Diffusion simulations naturally fit these characteristics and they have been implemented using GPGPU programming (Vigelius, Lane and Meyer, 2011).

3.3 Cloud Computing

The limits of using GPGPU implementations, however, are that they are restricted to the principle of SIMD parallelism. A solution for high performance simulations which combine the affordability and accessibility of GPUs with the MIMD architecture of supercomputers has led researchers to look at using cloud platforms. The development of supercomputers, clusters and grid computing makes the line between such hardware and cloud computing thin. Many of the problems that face

the former also face the latter and all such platforms can be broadly grouped under the umbrella of 'distributed systems' (Fujimoto, 2003). The distinction that is made for the purposes of this paper is that cloud platforms have no guarantee of having nodes in the same geographical location and so their synchronization and data distribution is at the mercy of network communication latency.

For a distributed system of any kind, communication time is a vital consideration. The distribution of data throughout a system is a requirement for any meaningful simulation and this is dependent on network communication. It has been observed that for a simulation, a scale and number of nodes can be used which has an acceptable cost of communication. However, as the number of simulated entities or nodes running the simulation increase, so too does the percentage cost of communication (Barrett *et al.,* Nov 15, 2008).

Synchronization is also an area of concern for any distributed system. Synchronization is required for a node to be sure that the data it is using for its processes is up to date and that it knows what order incoming messages should be in. Certain types of simulation are more sensitive to the accuracy of synchronisation. A sequential event driven simulation which progresses through the propagation of discrete events requires a tightly synchronised implementation (Lees, Logan and Theodoropoulos, June, 2003). If events are received in the wrong order or out of step, the flow of the simulation can be massively disrupted. Techniques such as conservative or optimistic synchronisation have been developed which handle the synchronisation issue on a distributed system (Dirksen, 2013). A simulation that is not event driven instead relies on continuous data updates. For this style of simulation, synchronisation is less of a concern due to the frequency of updates and the non-sequential nature of the simulation.

3.4 SpatialOS

Improbable has built a cloud computing platform for handling distributed simulations called SpatialOS (Franklin-Wallis, 2017). SpatialOS integrates with existing game engines whilst also providing support for constructing bespoke applications to be managed by the platform. Improbable have created multiple large-scale simulations which demonstrate the power of the SpatialOS platform. One of the first such simulations was the city of Manchester, UK, modelling traffic, energy and communications networks (Parkin, 2016). A later simulation models the infrastructure of the internet (Improbable, 2016). This demonstrates the potential for the platform to support large scale simulation, however a technical analysis of these simulations is not publicly available.

This technology only became available in an open alpha format for developers at the beginning of 2017 (Sunar, 2016). As such there is currently no studies which have been completed using SpatialOS for implementing a simulation. Learning from the successes and failures of other academic studies using the platform may encourage more researchers to utilise the potential power of the platform. Additionally, a performance analysis of the platform may provide a good starting point for other studies to build simulations which make best use of the specific design characteristics of SpatialOS.

3.5 Performance Analysis

When conducting a performance analysis of a system, a variety of profiling techniques can be used to obtain a large range of metrics. The metrics are selected based on the performance characteristics of the system and the data gathered in a manner suitable to the specific metric. Which specific metrics are selected, and whether the list is updated during the analysis, will depend on unique system needs and the researchers' experience with the system (Braddock, Claunch and Rainbolt, Mar 1, 1992).

In summary, the potential of cloud computing to dramatically alter how scientific simulations are conducted is huge. By using a platform such as SpatialOS, the affordability of SIMD systems such as GPGPUs can be combined with the more flexible range of functionality that MIMD systems, such as supercomputers, provide. Before researchers take up this new platform, however, evidence of its use need to be documented and analysed. Alongside this a set of best practices that relate to implementation specifics, beyond high-level design concepts, needs to be created. How this system will be analysed will rely upon the unique properties of different types of simulation, however there is still an evident need for some initial analysis to be conducted to provide researchers with a starting point for developing simulations which make best use of this new technology.

4. Methodology

This section describes the tools and techniques used to construct the application as well as the processes which were followed to construct experiments to obtain performance data from SpatialOS. Particular emphasis is placed on the structure and implementation of a SpatialOS project in general as the conceptual leaps and understanding required to do this effectively formed the most significant part of the development process.

4.1 Development Tools

4.1.1 SpatialOS

Given that the purpose of this project was to explore the implementation of a simulation running on the SpatialOS platform, heavy use was made of the development tools provided by Improbable. Of these tools the SpatialOS Unity Software Development Kit (SDK) and Command Line Interface (CLI) tool were the most heavily used as they are essential when using the platform for development and deployment of an application. The Unity SDK is available as a plugin and provides quick access to various aspects of the CLI tool through custom Unity editors, though certain functionality, such as deploying to the cloud, requires the use of the CLI tool.

The Improbable Console was used for debugging and visualisation of the deployment. The console was used for local and cloud deployments and has specific functionality depending on which of these targets has been deployed to. For local deployments, the console provides an inspector which visualises the simulated game world. The inspector visualises the spatial arrangement of all entities in world space as well as which workers currently have read and/or write access over the entities. Furthermore, the inspector displays a list of all active workers and clients which can be inspected to view the data concerning which entities they are interested in, can

see and have authority over. Individual entities can also be inspected to read the data governed by their SpatialOS components.

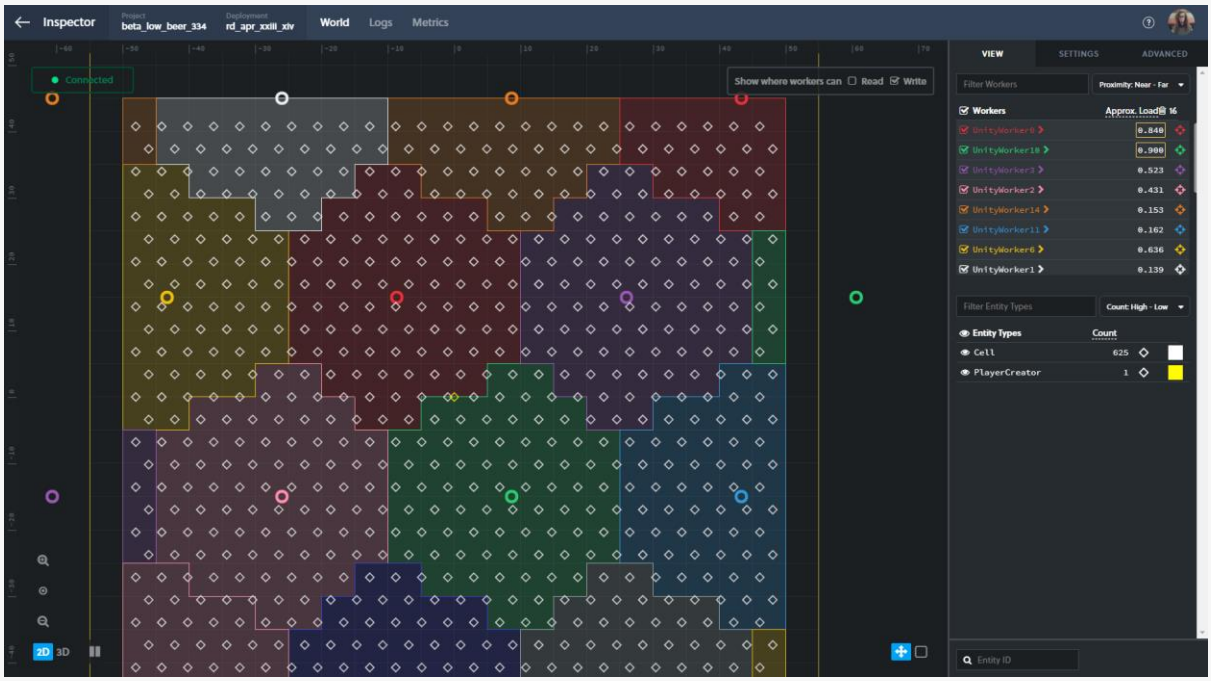


Figure 1. Improbable's Console Inspector viewing a deployment of 625 cells across 16 workers.

This functionality is vital in understanding how the application is behaving at runtime and provides a great starting point in the debugging process. For cloud deployments, the console provides access to other features in addition to the inspector such as up-time, the ability to launch a client application and the ability to save the state of the simulation as a snapshot. There are also further tools which help with debugging that are unique to cloud deployments, such as the ability to view a collection of the real-time messages being logged by the workers, access to the raw verbose logs of each worker, and a portal to the metrics system which provides access to further runtime performance data.

The metrics system provided by Improbable is a Grafana implementation that provides numerous dashboards containing performance graphs revealing information about how various aspects of the deployment are performing. As this project is concerned with analysing the performance of a simulation running on

SpatialOS the metrics system was used heavily in debugging, optimisation and results gathering.

4.1.2 Unity

SpatialOS currently has support for a variety of development environments, such as C#, C++, Java, Unreal Engine and Unity. A simulation running on SpatialOS can make use of different worker types developed in different environments which interact with each other through the interface of SpatialOS components. For example, the client may be developed using Unity to make use of the pre-existing rendering or user interface systems, whilst the AI system could be developed using C++ as this system's functionality is independent of much of Unity's functionality. The workers are able to see the same entities and the SpatialOS components on those entities which they are interested in, therefore governing different aspects the same simulation.

Due to the time-scale of this project it was decided that a single development environment would be used, so that a focus could be placed on learning the API of a single SDK. For two main reasons, Unity was selected. Firstly, the pre-existing framework for rendering, UI, scene management and physics meant that a game engine was a sensible choice as the development of these systems were outside the scope of the project. Secondly, Unity was chosen over Unreal due to the maturity of the SDK. SpatialOS is a new technology which is undergoing active development, the various SDKs are also undergoing active development and the Unity SDK was the first to be created. This meant that at the commencement of the project the Unity SDK was far more stable than the Unreal SDK and the project was less likely to face setbacks due to bugs/missing functionality in the SDK.

4.1.3 VS Code

Development for SpatialOS using Unity requires the reading and writing of various types of code and data. VS Code was used as a coding environment due to its great syntax highlighting and code exploration for C# and .json files. A plugin was also

used which provides syntax highlighting for Improbable's Schemalang language which is used for the construction of SpatialOS components. Furthermore, it provides easy to read automatic formatting of the log files output by workers and the spatial CLI.

4.2 Development Process

Due to the exploratory nature of this project and the young age of the SpatialOS platform, an agile approach with frequent prototyping was essential. Simulation prototypes were built using blank Unity projects so that their algorithms and functionality could be tested and understood before attempting to integrate them with SpatialOS. Once expected behaviour was achieved with these prototypes, they were ported into a SpatialOS Unity project with the aim of achieving a local deployment running on a single worker on the development machine. The same simulation was then deployed to a single worker on the cloud, to ensure that it behaved in the same way when the development machine was only running the client application. Following this the implementation was modified to run on multiple workers using a local deployment on the development machine. Up until this point the simulation parameters, such as number of entities, had to be continually reduced as more complexity and load was being added to the single machine that was running the simulation. The final step in the simulation development was to then test the deployment on the cloud utilising multiple workers. Once this stage had been reached performance analysis and optimisation could begin.

This part of the project lifecycle was heavily focused on generating prototypes to understand the nature of SpatialOS which could provide context and leads for the optimisation and analysis stage which would follow. The second phase of the project could follow a more 'standard' approach to agile development.

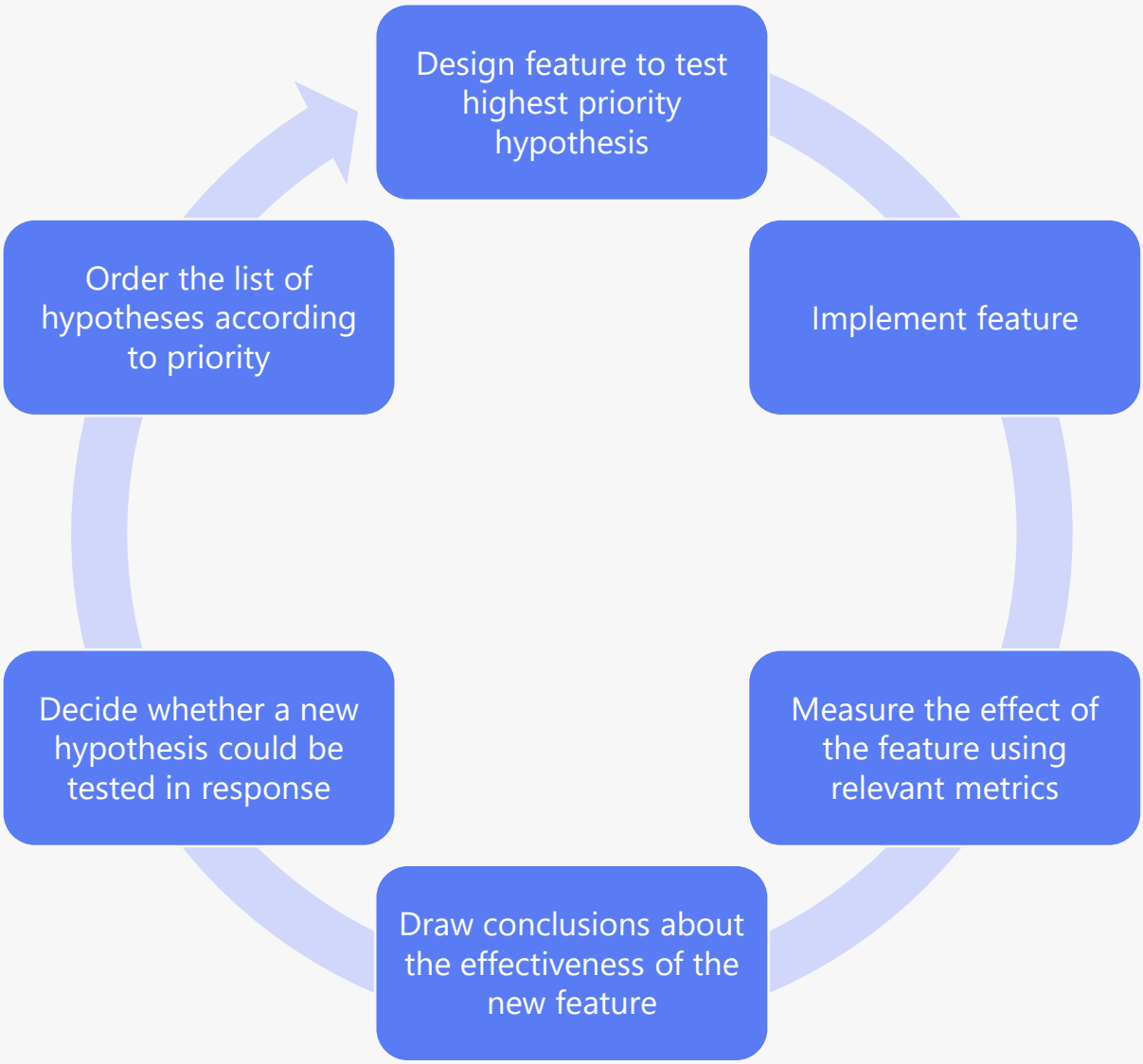


Figure 2. A diagram which shows the process of agile development.

A hypothesis was devised which would question whether a technique or configuration could improve performance. The metrics by which this hypothesis could be tested was decided upon and relevant measurements were taken. The technique, configuration or feature was then implemented and tested again to see how the measurements had varied. The results of this test were then considered and the next hypothesis was decided upon and designed. The main benefit of this approach is that the list of proposed hypotheses is regularly re-prioritised allowing emergent discoveries to be acted upon quickly.

4.3 Feasibility Demo

A feasibility demo was produced to gain an understanding of the functionality of SpatialOS which could inform the rest of the project. For this demo a simulation model was chosen according to a set of desirable characteristics. The first characteristic was that the simulation be comprised of entities of a single type, whose interactions with one another exhibit observably interesting patterns. Since the purpose of the project was to analyse the performance of a simulation it was unnecessary to spend a significant amount of development time on creating a behaviourally complex simulation. This leads to the second characteristic that the entities' functionality be computationally complex. The project's requirements are to test the performance of the platform, so the simulation must be able to stress-test the systems to some extent, beyond the potential bottleneck of number of entities able to be rendered. The third characteristic, in slight contrast to the previous, was that the entities behaviour should be relatively simple to implement. There is no need for the project to have advanced artificial intelligence techniques managing autonomous agents, mathematical or logical complexity would suffice.

These characteristics were decided upon without a deep knowledge of how SpatialOS was structured. The purpose of the feasibility demo was to gain that knowledge, so it was expected that the requirements of the simulation may change further into the project.

The simulation used for this demonstration was a brute force Newtonian Gravity N-Body simulation (Hut and Barnes, 1986). Each entity was a body with mass, the body would calculate the sum of the gravitational forces enacted upon it by all the other bodies, due to their position and mass. The resultant force would then be applied and the body's position would be updated according to its new velocity. This had all the required characteristics as well as some other helpful features, such as its potential to have interactivity, the potential to dynamically scale the simulation and

the ability to visualise it in the Improbable Console rather than just through rendering it on a client application.

The simulation was implemented up to the point of running it on multiple workers in the cloud. At this point, it was observed that a key design requirement of a SpatialOS simulation is that it should be spatially local. The gravity simulation is global at its core, the gravity of every body must affect every other body. This is not suitable for a distributed system as the network communication required is not able to relay the information between workers at the required speed for a scientifically accurate simulation.

Modifications could have been made to expand upon this simulation and make it more suitable for SpatialOS. The framerate could have been reduced to account for the network load of this cross-communication. A more optimised form such as the Barnes-Hut tree algorithm could have been used to reduce the amount of network communication. Ultimately, however, the simulation would not be scalable as the complexity for the brute force is $O(n^2)$ and even the Barnes-Hut is $O(n \log n)$ (Hut and Barnes, 1986).

4.4 Simulation Design

A different simulation model was selected for the continuation of the project, with the additional requirement that each entity perform its behaviour in some spatial locality to be better suited for the SpatialOS platform. A simulation which had this characteristic, in addition to the original characteristics, is the Reaction Diffusion simulation. This simulation's Dwarf categorisation is a structured grid, in comparison to the N-Body type used for the feasibility demo (Asanovic *et al.*, 2006). The Reaction Diffusion simulation has two chemicals: A and B. The simulation space is divided into cells and the concentrations of A and B are calculated as they vary over the space. Each cell calculates the concentration of both A and B within it according to the

concentrations of the two in neighbouring cells, as well as some global simulation parameters which can be defined upon initialisation.

4.4.1 Theoretical Model of a Reaction Diffusion Simulation

The simulation is governed by a pair of equations, one for each chemical, which are calculated in each cell every frame (Sims, 2013).

$$A' = A + (D_A \nabla^2 A - AB^2 + f(1 - A))\Delta t$$

$$B' = B + (D_B \nabla^2 B + AB^2 - (k + f)B)\Delta t$$

$[A'], [B']$: The new concentrations of chemical A and chemical B in the cell.

$[A], [B]$: The previous concentrations of chemical A and chemical B in the cell.

$[D_A], [D_B]$: The diffusion rates for chemicals A and B

$[\nabla^2 A], [\nabla^2 B]$: The contribution of the concentrations of A and B in the surrounding cells. These are Laplacian functions which calculate the difference between the average of nearby cells and this cell. This is responsible for the diffusion effect of the simulation.

$[AB^2]$: One particle of A and two particles of B react with each other to create a single particle of B ($A \times B \times B$). This is taken away from the new concentration of A and added to the new concentration of B. This is responsible for the reaction effect of the simulation.

$[f]$: The feed rate of A into the system. A is added to each cell in an inversely proportional amount to its current concentration of A. The feed rate prevents A from being removed entirely from the simulation by the reaction process.

$[k]$: The kill rate of B from the equation. B is removed from each cell in a proportional amount to its current concentration of B. The kill rate prevents an insurmountable build-up of B in the system due to the reaction process.

$[\Delta t]$: The delta time. All changes to the concentration are scaled by delta time to control the speed at which the simulation is performed.

For the Laplacian functions a 3x3 convolution matrix was used. The concentration of the chemical is scaled by the concentration of the same chemical in surrounding cells and the inverse of the concentration in the cell performing the calculation. The following scale values are used for the 8 cells surrounding and the cell itself:

0.05	0.2	0.05
0.2	-1	0.2
0.05	0.2	0.05

4.4.2 Reaction Diffusion Algorithm

The simulation calculations are performed inside a `DiffusionCell` class. In the Update function, the cell calculates its new concentration values based on the concentrations of the chemicals in the surrounding cells. In the late update function this new value is then applied to the cell to maintain synchronisation across the simulation.

```
void Calculate(Neighbors<CellProperties> neighbors)
{
    double delta = Time.deltaTime;

    double reaction = cell.Aconcentration * cell.Bconcentration * cell.Bconcentration;

    double f = feed * (1.0 - cell.Aconcentration);
    double k = (kill + feed) * cell.Bconcentration;

    double da = aDiffusion * LaplacianA(neighbors);
    double db = bDiffusion * LaplacianB(neighbors);

    double newA = cell.Aconcentration + ((da - reaction + f) * delta);
    double newB = cell.Bconcentration + ((db + reaction - k) * delta);

    SendPropertiesToSpatial(newA, newB);
}
```

Listing 1. A code snippet which shows the `Calculate` method from the `DiffusionCell` class which performs the reaction diffusion calculations.

4.4.3 Neighbour Searching Algorithm

The Reaction Diffusion simulation model is much better suited for the SpatialOS platform than the N-Body simulation due to a cell only being dependant on its neighbouring cells. The amount of locality can be modified by using a larger convolution matrix that uses neighbours at a greater distance away, but the simulation does not need to consider every other cell. A data structure which holds references to each cell is not a viable concept for SpatialOS and a potentially infinitely large simulation. Instead each cell needs to be responsible for finding and referencing its own neighbours. All cells have a Unity physics collider attached to them. A cell calls the Unity physics overlap sphere method which then returns an array of all cells in the defined locality. These cells are then sorted in order of their placement around the central cell so that the correct scaling value is applied. Using the expensive physics method for neighbour searching each frame was unnecessary if the grid was static and the neighbours did not change. Neighbours were discovered during the initialisation of the simulation and stored locally in a Unity component. When ownership of the cell is transferred, these references are lost so it was still necessary to occasionally repeat the algorithm in response to authority changes.

4.4.4 Expandability of the Simulation

The complexity of the simulation is expected to be linear on one worker, $O(n \cdot c)$ where n is number of cells and c is the chosen constant number of neighbours, as a cell's behaviour and calculation time is not affected by the number of other cells, only the number of neighbours. This makes it far more scalable than the N-Body model as the simulation can be expanded onto more cloud resources in direct correlation to the number of entities being simulated.

4.5 SpatialOS Design Considerations

4.5.1 Worker Types, Authority and Component Visibility

Entities in SpatialOS are made up of a mixture of Unity components and SpatialOS components. A SpatialOS component is written in Improbable's Schemalang language. This basically provides a shortcut for developers to write the basic characteristics of the component which is used by the spatial code generator to create comprehensive C# classes, containing lots of boilerplate code, which can be interacted with through regular Unity scripts. The schemalang component is comprised of command declarations and variables of system or user defined types. All components have a unique id member which is used internally by SpatialOS.

```
type ResetRequest
{
  double k = 1;
  double f = 2;
  double ad = 3;
  double bd = 4;
  double dt = 5;
}

type ResetResponse {}

component Concentrations
{
  id = 1005;
  double a = 1;
  double b = 2;
  bool is_resetting = 3;
  float timestamp = 4;
  command ResetResponse reset_values(ResetRequest);
}
```

Listing 2. A code snippet which shows the Schemalang declaration for the `Concentrations` SpatialOS component which is used to generate the `Concentrations` C# class.

Worker types can be given the ability to read from components or read from and write to components, for example the position data for a cell can only be written to

by a physics worker but can be read by both the client and physics worker. Only one physics worker has the authority to write to this component at any time, but any physics or client worker whose domain of interest this cell is in can read from the component. When a worker has write access to a SpatialOS component is has a `Component.Writer` object. This can be required by a Unity component which allows visibility of a Unity component to be restricted to only the worker which has write access to the SpatialOS component. Likewise, the `Component.Reader` object is given to any worker with read access allowing entities to have different levels of Unity component visibility depending on whether the viewer has read access, write access or no access to specific components attached to the entity.

Write Access	**SpatialOS Component**	**Component's Data and Commands**
Physics & Client	`Initialiser`	`a` : Initial concentration of A `b` : Initial concentration of B `k` : Simulation kill rate `f` : Simulation feed rate `ad` : Diffusion rate for A `bd` : Diffusion rate for B `dt` : The simulation rate modifier `reset_cell` : Whether to reset the cell values
Physics	`Concentrations`	`a` : Current concentration of A `b` : Current concentration of B `is_resetting` : whether the cell is resetting or running `timestamp` : The system time of the update `reset_values()`: Command which resets simulation variables
	`CellSize`	`size` : The physical size of the cell

Table 1. A table which displays the user defined member variables of the SpatialOS components attached to a cell entity in addition to which worker types can have write access to each component.

Workers with Access	Unity Component	Authority Requirements
Client	`CellVisuals`	`Concentrations.Reader` `CellSize.Reader`
	`CellSync`	`Concentrations.Reader` `Initialiser.Writer`
Physics	`CellProperties`	`Initialiser.Reader` `Concentrations.Reader`

	`DiffusionCell`	`Initialiser.Reader`
		`Concentrations.Writer`
		`CellSize.Writer`

	`NeighbourSearching`	`Concentrations.Writer` `CellSize.Writer`

Table 2. A table which displays the Unity components attached to a cell entity in addition to which workers they can be enabled on, and what read or write access is required to enable the script.

Two types of worker were created for this simulation: a client and a physics worker. The client is not responsible for any of the simulation calculations; it is only responsible for visually displaying the cells, so that the behaviour can be observed, and for interacting with the simulation. For this project interaction is limited to the adjusting to simulation parameters, stopping and restarting the simulation which is done through a UI attached to the client. To perform this functionality, the client needs to have read access to all entities that are in view of the camera regardless of which physics worker has authority over each individual cell. Specifically, the client needs to be able to read the concentrations data which is updated frame by frame by the physics workers. Additional cell functionality, that is unique to the client, is required for visualisation. The cells have a `CellVisuals` script attached to them which update the shader rendering each cell, the client performs the functionality of this visuals script which uses the cell's concentration values to update its colour

depending on the proportion of the two chemicals to one another in the cell. The client is also responsible for spawning a player entity into the simulation space.

The physics workers perform the neighbour searching and the simulation calculations. They do not need to know about the visualisation of the cells and so do not have access to these scripts. The calculations are only carried out by one worker, although all physics workers have the potential to carry out the calculations on any cell. The concentrations data of the cells is stored in a Unity component called `CellProperties`, which can be read by all physics workers allowing for data to be read from neighbouring cells. The calculations are executed in the `DiffusionCell` script which is only active on the worker with write access to the `Concentrations` SpatialOS Component.

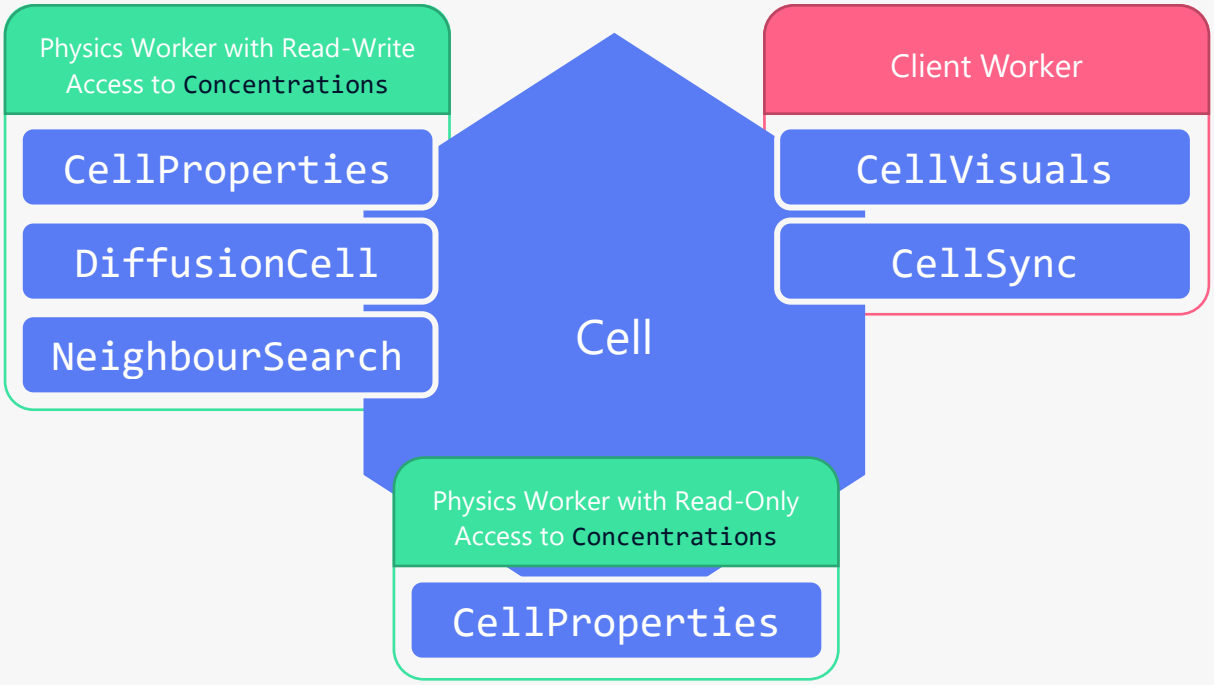


Figure 3. A screenshot of the client application viewing a simulation of 1600 entities with a square grid layout after a point of equilibrium had been reached.

4.5.2 Network Considerations for a Distributed Simulation

Workers in SpatialOS could potentially die or be started up at any time. This allows the platform to effectively load balance and respond to unexpected hardware malfunction without significantly disrupting the simulated world. This means,

however, that any data stored locally on a Unity component at the time of a worker death is lost. In some cases, this may be acceptable, however if this data is important for the simulation it needs to be stored elsewhere to prevent unexpected data loss. The solution for this is to store the data in SpatialOS components. The data in a SpatialOS component attached to a given entity is stored independently of the worker which has authority over that entity, so if the worker goes offline another worker can gain authority over the entity and has the correct data in the SpatialOS component. This means that data needs to be sent back to SpatialOS from the Unity components as a component update every frame. In addition to this, Unity component data needs to be requested from the SpatialOS component every frame, as the entity may have recently transferred to another worker's authority resulting in its local Unity component data to be reset. Performing the component update and component data requests every frame for each cell, and then again performing the component data request on each cell for every worker that can see it, relies heavily on network communications. Limiting the amount of times that these operations are performed is an important design consideration when constructing the logic of the simulation.

The network dependence of SpatialOS also affects how the neighbour searching is performed. Objects can be searched for by either using a SpatialOS query or by using a Unity method. SpatialOS queries are very expensive in terms of network usage, but can return references to entities that may not be in a worker's area of interest. Unity methods, such as `FindGameObject`, are not reliant on network communication and so can be performed far more frequently. The issue with this type of method, however, is that it will only return entities which are in a worker's area of interest. This problem can be overcome by careful consideration of the overall simulation design, it must be constructed so that the neighbour search radius will always fall within the area of interest for a worker if the worker is authoritative over that entity.

4.5.3 Launch Configuration and Load Balancing

The name of the platform 'SpatialOS' describes its unique way of functioning. All distribution is performed spatially, the workers are authoritative over a specific area in physical space. How this domain of authority is calculated, is governed by the launch configuration of the deployment. There are some key parameters in the launch configuration .json file which may be adjusted to change the way the SpatialOS platform handles the spatial arrangement and authority distribution of workers running the simulation.

Fundamentally, the world simulation space parameters must be set up which puts a maximum size on the area that the simulation occurs in and how this space is subdivided. The dimensions of the space are set in terms of meters in the x,z plane. This space is then divided into chunks of size `chunkEdgeLengthMetres`, which must be a factor of both the x and z dimensions. If a worker is authoritative over an entity in a given chunk, the worker's domain of interest also extends into all the neighbouring chunks and it has read access to components in these chunks. For this project it is imperative that the chunk size is at least equal to the neighbour search radius so that a worker will always be able to read the concentrations of all neighbouring cells to a cell that it is authoritative over.

Setting up the workers' spatial arrangement and load balancing uses a more complex system of parameters. At its base level, the configuration can be set to static or dynamic allocation of workers. A static configuration exists in a hexagonal grid layout where once workers have been created at the beginning of the deployment their domain and position do not change, regardless of whether they are under heavy or light load. This hexagonal grid can either be automatically created by SpatialOS, or be given a number of rows / columns with a radius of interest for the workers. Either way, the hexagonal grid must cover the entire simulated world space.

A dynamic configuration allows workers to be moved, created or removed in response to load. This is split into three sub-configurations. `worker_scaler_config`

determines whether the number of workers is constant or changes according to the number of entities. For a simulation with a constant number of entities, it makes more sense to set the number of workers as constant. The developer account used for this project limited the number of resources permitted to be used to 25, so this was the maximum number which was set. `worker_placer_config` defines whether the workers are randomly placed or adhere to a hexagonal grid.

```
"worker_type": "UnityWorker",
  "load_balancing": {
    "dynamic_loadbalancer": {
      "worker_scaler_config": {
        "constant_config": {
          "num_workers": 25
        }
      },
      "worker_placer_config": {
        "random_params": {}
      },
      "loadbalancer_config": {
        "min_range_meters": 4.0,
        "max_range_meters": 28.0,
        "speed_meters_per_second": 2.0,
        "expansion_time_millis": 6000
      }
    }
  }
```

Listing 3. A code snippet from the launch configuration file which shows the load balancing declaration. The configuration displayed is that of 25 workers arranged randomly according to load.

The random configuration allows workers to be positioned arbitrarily in response to areas of high load. The hexagonal grid parameters are set up in a similar way to that of the static configuration, allowing for automatic or manual specifications of grid dimensions and worker radii. When used as a grid, this sets an upper limit on the configuration, but allows resources to be freed up if the simulation is not experiencing a heavy load. Finally, `loadbalancer_config` sets how large or small the worker's domain can be, how quickly it can change size and how quickly it can move

position in response to load. These parameters are used to make sure that workers can be distributed to cover the full simulation space, and to stabilise the amount of authority changes that occur over time.

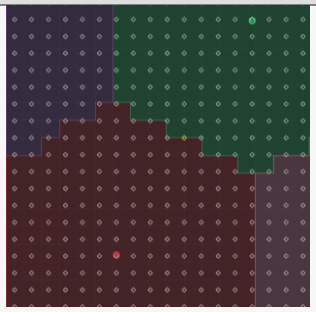
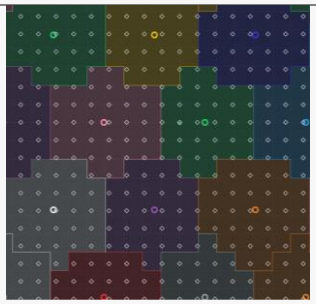
Launch Configuration	Number of Workers	Spatial Arrangement of Workers
Static Automatic Hexagonal Grid	4	
	25	
Dynamic Random Arrangement	25	

Table 3. A table showing the changing layout of workers when launched according to different load balancing parameters.

As the amount of computation that occurred each frame was constant in this project, and the entities positions were constant, a static configuration was most suitable. However, dynamic configurations were also investigated to understand how the configurations should be changed in response to runtime entity creation or different densities of computational complexity across the simulation space.

4.6 Design Considerations and Optimisations

The project's aims were to analyse the performance of SpatialOS when running a complex simulation, and how this performance is affected by adjusting the design of the systems used for the simulation. To optimize the systems, performance bottlenecks were identified and adjustments were made to the implementation in an agile way so that the effects of the optimisations could be quantified.

Knowing whether all neighbouring cells are under the same worker's ownership allows for a major avenue of optimisation. The network load of sending the concentrations of the cell to the SpatialOS component every frame could be reduced if the surrounding cells could access that data directly without requesting from SpatialOS. This can only occur if a cell knows that its neighbours are all on the same worker. This also means that the data does not need to be sent to the SpatialOS component every frame, only cells on the periphery of a worker's domain would need to send and receive regular updates.

Another optimisation that was implemented was the modification of the grid type and cell shape to provide different layouts and different neighbour relationships. Initially, the cells were square and had four major neighbours (sharing an edge) with four minor neighbours (sharing a vertex). This meant that the neighbours needed to be spatially sorted so that the Laplacian function could be applied correctly and the weightings of the convolution matrix accurately represented the diffusion potential of the neighbour. In nature cells will often form a hexagonal structure, such as a honeybee's honeycomb. This is due to the hexagonal grid offering least resistance with a minimal amount of shared surface area and minimising free energy (Ball, 2009).

This structure had two potential grounds for optimisation. Firstly, the number of neighbours would be reduced from 8 to 6 minorly reducing the amount of neighbour references stored and reducing the number of neighbours to iterate

through for the Laplacian function. This served as a communications optimisation as the cells would each need to communicate with 25% fewer neighbours within the same locality radius. Also, each of the 6 neighbours of a hexagonal cell shares the same number of edges and vertices with the central cell. This negates the need for a convolution matrix or any sorting of the neighbours at all. The neighbours would all get an equal concentration scaling factor of (1/6). This was a physics optimisation and sought to reduce the load per entity of the implementation on the physics workers.

4.7 Evaluation Methods

4.7.1 Metrics

The metrics output by the SpatialOS deployment were used to measure the baseline performance of the project and to quantify the effects of the implemented optimisations. Specific metrics were identified which would serve as key performance indicators (KPI's) for the performance of the simulation in several areas. The simulation performance was observed by looking at the composite worker load, how many workers would ideally be being used to run the simulation with its current parameters. Additionally, the average framerates across the workers was used to identify whether the simulation was reaching its target framerate on the workers and therefore whether the workers are under too much load. The total network egress and ingress was used to measure the amount that the implementation was communicating with SpatialOS. The number of messages that were sent between workers and SpatialOS was also tracked along with the composition of these messages. As this provides insight on what specific aspects of the implementation were dominating the network.

4.7.2 Client Visualisation

The client was used to visualise the concentration levels of the cells and therefore to observe the behaviour of the simulation. For each cell, the concentrations of A and B

were added together and a value was obtained that described the amount of A proportional to the total amount of each chemical:

$$t = \frac{A}{A + B}$$

This value was used by the shader to colour the cell according to a defined palette. For any given value of t, the colour used is a linear interpolation between the two colours of the thresholds t sits between. The simulation's behaviour was assessed using face validity to see whether the predictable patterns that were expected were what was being observed.

Colour											
Threshold	0.0	0.2	0.3	0.35	0.4	0.433	0.466	0.5	0.55	0.7	1.0

Table 4. A table which shows the threshold values of chemical A as a portion of total chemical volume, as well as the colour value associated with that threshold value.

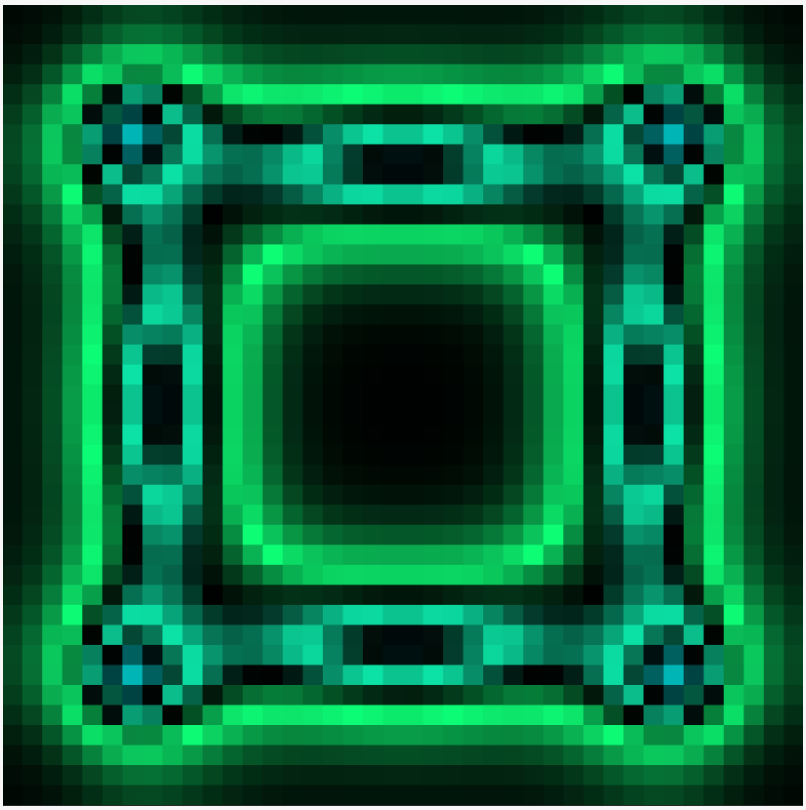


Figure 4. A screenshot of the client application viewing a simulation of 1600 entities with a square grid layout after a point of equilibrium had been reached.

5. Results

Data was collected by using the Improbable Metrics tool, which provides real-time data across a plethora of categories and individual metrics. The data collected came in the form of graphs which show the results per-metric and per experiment. The original graphs captured from the metrics system can be viewed in the appendix. In the original format it is very difficult to visualise a comparison between different experiments, due to the sheer quantity of data. This comparison is necessary to draw conclusions about the performance according to a particular metric across different scales of deployment and simulation. The data was aggregated and graphs were created which display a single metric and how each experiment performed across that metric.

5.1 Worker Load Balancing

The following figures are taken from the Improbable Console tool. The white diamonds represent individual cells. The coloured regions indicate each individual worker's domain of write access. As the number of workers increases, the domain of write access decreases. The coloured circles represent the 'centre of mass' of the worker whose domain is the same colour. The experiments used for these figures are the different deployment sizes of a hexagonal layout simulation with 900 cells.

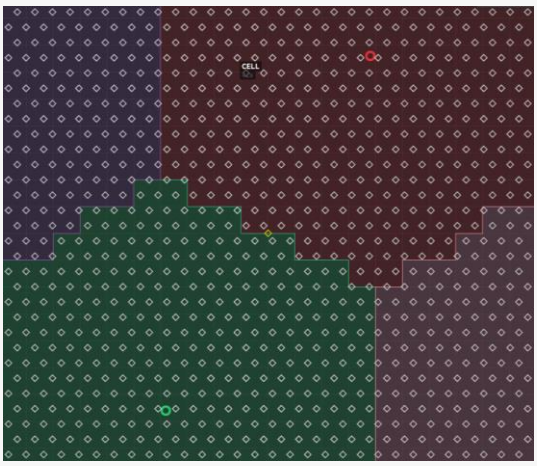


Figure 5. A screenshot showing the spatial

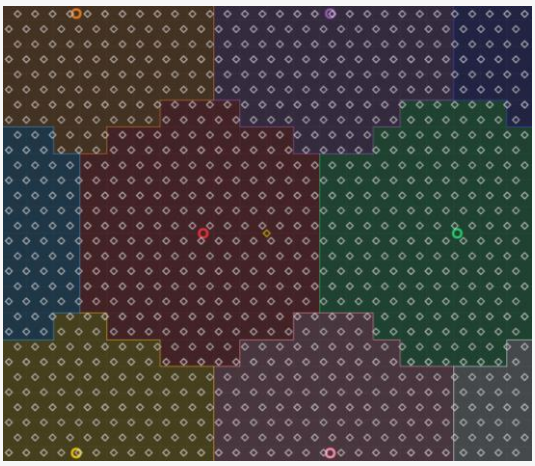


Figure 6. A screenshot showing the spatial

arrangement of 4 workers simulating 900 cells in a hexagonal layout of cells and workers.

Figure 7. A screenshot showing the spatial arrangement of 16 workers simulating 900 cells in a hexagonal layout of cells and workers.

arrangement of 9 workers simulating 900 cells in a hexagonal layout of cells and workers.

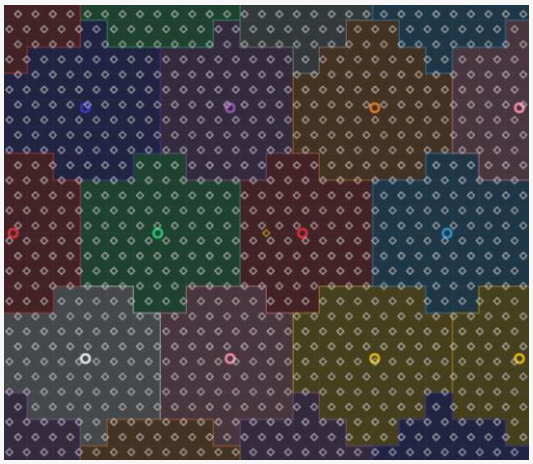


Figure 8. A screenshot showing the spatial arrangement of 25 workers simulating 900 cells in a hexagonal layout of cells and workers.

5.2 Performance as Number of Workers Increases

The following graphs give an overview of the performance of the simulation as the number of workers running the simulation (the scale of the deployment) increases. For each of the graphs in this section, the pink line represents the base implementation of the simulation with a regular square grid layout and no network communication optimisation. The blue lines represent the square grid layout with network communication only happening every frame at the border of the workers' domains. The green lines represent the same network communication optimisation, but on a hexagonal grid. For each of the blue and green lines, the darker the colour the larger the scale of the simulation (larger number of cells).

Colour Used for Graphs	Number of Cells	Simulation Configuration
	441	Square grid
	441	Square grid with communications optimisation
	625	
	900	
	1600	
	441	Hexagonal grid with communications optimisation
	625	
	900	
	1600	

Table 5. The legend for each of the graphs presented in the results showing the bar colour and experimental conditions for each deployment.

5.2.1 Worker Framerate

The value used for the average framerate is taken from an average between the lowest and highest framerate (in fps) of any of the workers currently active.

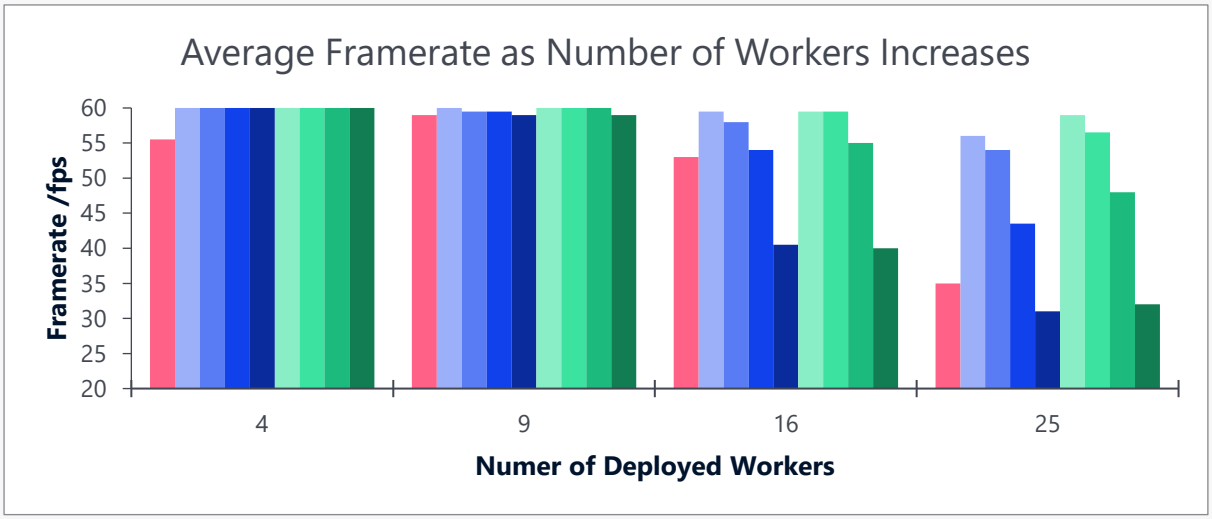


Figure 10. A graph which shows the average framerate of the workers for each experiment as the number of workers and number of entities increases.

5.2.2 Maximum Worker Load

The worker load displayed in Figure 9 is a number which relates to how overloaded the workers are. The data used to construct the graph is taken as an average over the duration of the simulation up-time of the worker load experienced by the worker under the highest load. For a given experiment and at a given time the max worker load is the number of workers which would ideally be handling the work-load of the worker under the highest load. This metric is useful for understanding the performance of the 'weakest-link' in the network and gives insight into how well load-balanced the simulation is. Ideally, the max worker load should not be above 1.0. Note the $\log_2 n$ scale for worker load.

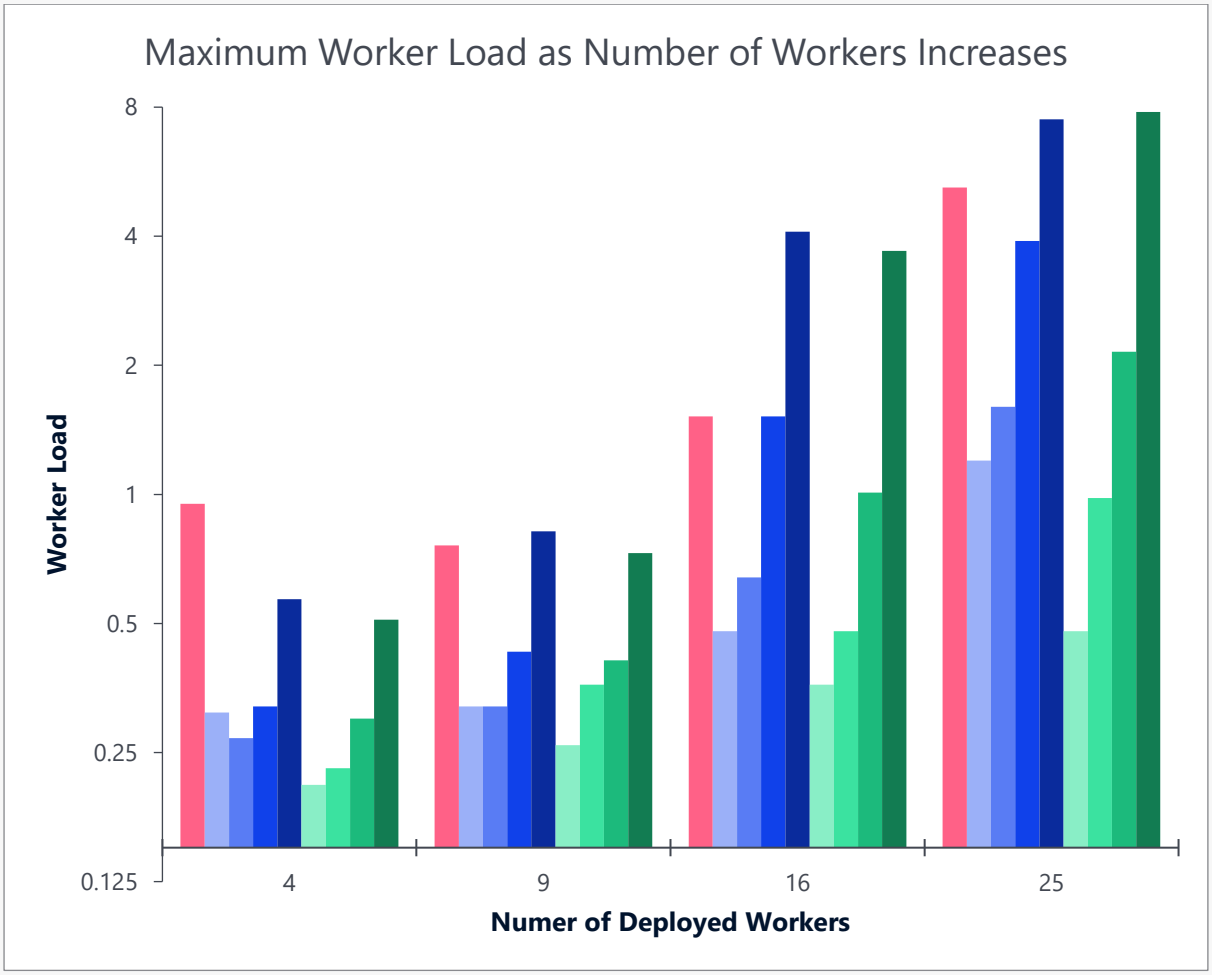


Figure 9. A graph which shows the average load of the worker under most load for each experiment as the number of workers and number of entities increases.

5.2.3 Worker Network Usage

The total network usage is split between amount of data received by SpatialOS from workers and amount of data sent to workers by SpatialOS.

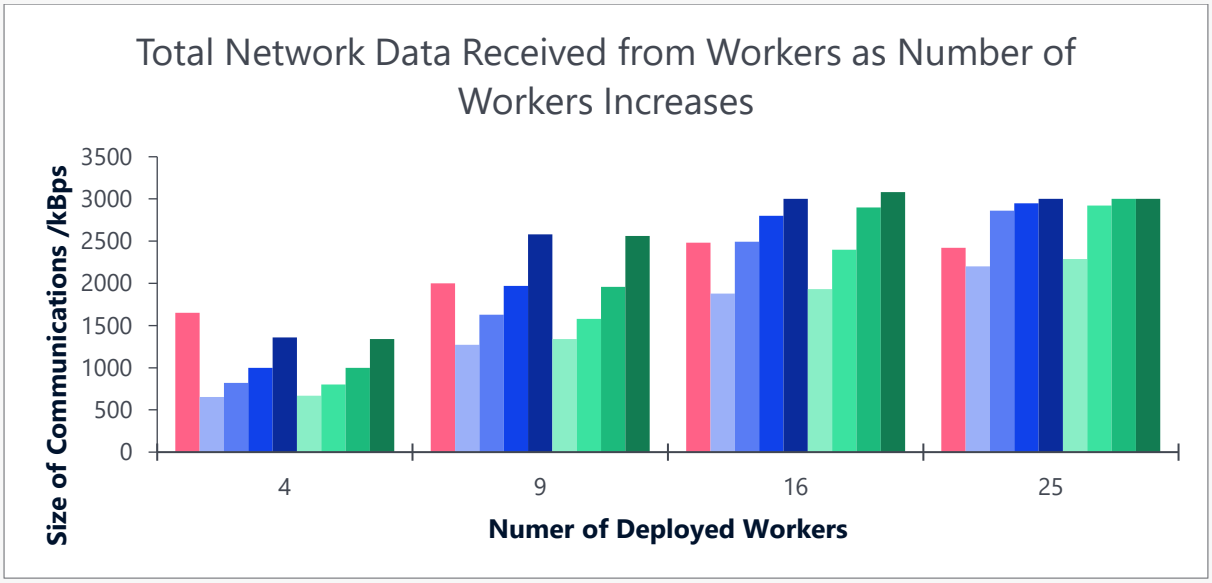


Figure 11. A graph which shows the rate of messages sent from workers to SpatialOS for each experiment as the number of workers and number of entities increases.

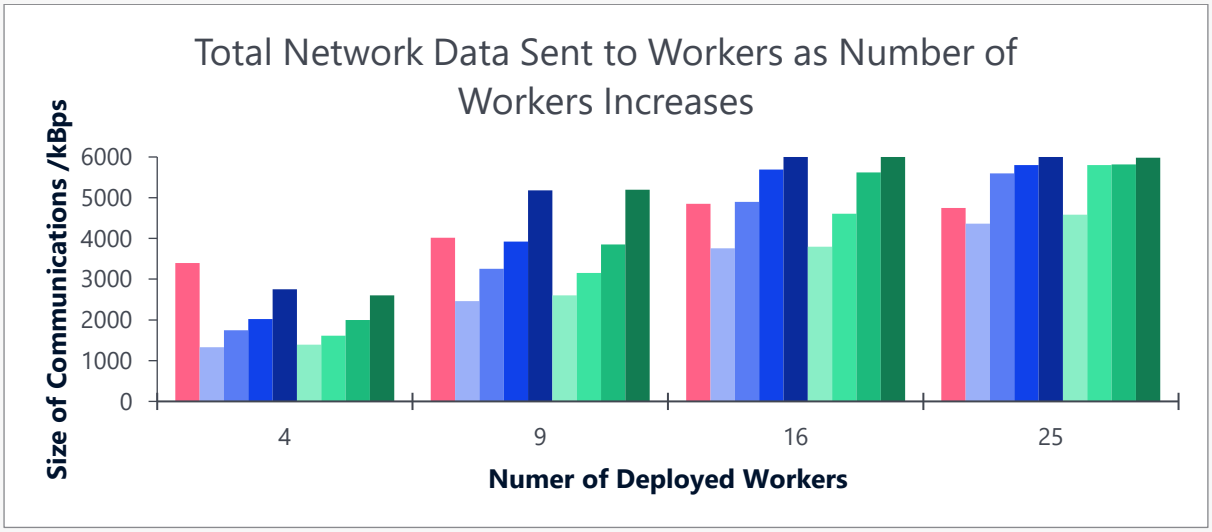


Figure 12. A graph which shows the rate of messages sent from SpatialOS to workers for each experiment as the number of workers and number of entities increases.

6. Discussion

This section addresses the results of the experiments conducted and constructs a performance profile of SpatialOS using the obtained data.

6.1 Worker Load Balancing

It can be seen from Figures 5 - 8 that the number of workers should not necessarily be used as a numerically accurate scale. If the conditions of the experiment were such that the number of cells were split equally between each of the workers, it would be possible to make accurate estimates about the effect of increasing the number of workers managing the simulation. For this paper, however, it will only be possible to make assumptions based on general trend. The reason for this is the sub-optimal layout of the workers. Specifically, looking at Figure 8, 25 deployed workers only equates to 20 workers managing any entities, and of those that are, the number of entities per worker varies dramatically. This is the case with all the different deployment worker counts, and is very difficult to quantify in a way that allows for assumptions to be made about how this scales. It is also unclear about whether the workers which are not managing entities are still impacting the performance or whether they can be ignored, for example whether they are still putting load on the SpatialOS platform by requesting updates or even sending their own internal updates.

The source of this sub-optimal layout is believed to be due to the auto-hex selection of the launch configuration, the reasons for choosing this parameter are outlined in the Methodology. If a method can be found to stabilise the launch configuration when using a random layout rather than a static layout, the SpatialOS load balancing algorithm could help to eliminate this uncertainty. Aside from this hard-to-define behaviour, the results obtained can still be considered reliable for the purposes of understanding the general trend of performance as the number of workers is increased.

6.2 Scaling Number of Workers

The effect of increasing the number of workers was expected to be an overall increase in the performance for the simulation. This was not what was observed and it is possible to explain why the initial assumption was incorrect. As the number of workers increases, the average number of entities per worker decreases. This means that the amount of computational power to process each of the workers' domains decreases. If this computational power requirement was the bottleneck (as was expected), an increase in performance of the individual workers would be observed as more workers were added to the solution. As the opposite effect is observed, the bottleneck must have to do with cross-worker communication. This was identified early in the results gathering phase as can be seen by the red bar in Figure 9 as well as in the Figure 10, which shows the effect of worker load on processing speed.

According to the development process of agile development and rapid prototyping, a solution was developed which sought to reduce the amount of communication per worker. The solution meant that communications would only happen on a frame-by-frame basis if the cell was at the edge of the workers domain and had at least one neighbour which was managed by a different worker. The result of this improvement was as much as a 38% increase in average worker framerate for 441 cells simulated across 25 workers. Even with this optimisation, the increase in network communication as the number of workers increased was the performance bottleneck. Increasing the number of workers increases the surface area of the combined circumferences of the worker's domains. This results in more cells having neighbours which are on other workers and therefore increases the number of cells which need to send and receive updates from SpatialOS each frame. The complexity of the simulation therefore increases with the number of workers, possibly to $O(n \cdot c \cdot 2^w)$ where n is number of cells, c is the chosen constant number of neighbours and w is the number of workers, although this is hard to define due to the previously addressed load balancing issues impacting the data.

If this trend were to be extrapolated to its limit where each cell was managed by a different worker, the optimisation would have no effect as every neighbour would be on a different worker. The following logical deduction can be made: if every cell is managed by a different worker, the communication is the absolute bottleneck and the processing of the cell's behaviour is inconsequential; if every cell is on the same worker, the processing of the cell's behaviour is the absolute bottleneck and the communication is inconsequential; therefore there must be a ratio specific to the implementation that defines what the ideal number of cells per worker is such that neither is the outstanding bottleneck. Determining this ratio is application specific and relies on the size of the neighbourhood as well as the complexity of the cell's behaviour. Insufficient data was collected to calculate exactly what this ratio would be for this project. Once the ratio is calculated, however, the number of workers deployed to the project can be defined as `entities_per_worker` allowing the simulation to dynamically change in size and adjust to maintain performance levels at scale. This would also require the previously suggested stabilised random arrangement of workers in the launch configuration.

6.3 Scaling Simulation Size

Increasing the number of cells in the simulation was always expected to decrease the worker performance when deployed to a constant number of workers. This is the observed effect, however the reasons why this effect was observed are different to those anticipated. It was expected that an increase in number of cells on a worker would increase the amount of computational power required by the worker to perform the simulation. However as described in the previous section, the actual bottleneck observed is still due to the reliance on network communications. Increasing the number of cells in a space increases the density of cells. This results in more cells being on the border of a worker's domain and therefore the number of cells which must communicate between workers increases.

Looking at the results from the hexagonal grid layout provides evidence that this is the case. If processing the functionality of the cells was the performance bottleneck, there would be very little difference in the performance between the two layouts when running the same number of cells on the same number of workers. Whereas the observed effect is a lower maximum worker load when using a hexagonal layout, as shown in Figure 9. If the issue with performance was communication, we would expect this performance difference to relate to the difference in number of neighbours. A square cell has 8 neighbours whereas a hexagonal cell has 6, 25% fewer. This aligns with the performance observed in the different deployments.

6.4 Network Communication Limits

The communication bandwidth is expected to have a hard limit for almost any system. By looking at Figure 10 and Figure 11, it appears the limit was reached when the scale of the simulation and deployment were too large. To understand if this is what happened, it is expected that two hypotheses would be true. The first hypothesis is that the maximum amount of data sent or received by SpatialOS would be the same whether a hexagonal or square grid layout were used. This is observed to be true, both the 16 and 25 worker deployments reach a limit of about 3MBps data received by SpatialOS from workers and about 6MBps data sent to workers by SpatialOS for both the hexagonal and square grid layout. In fact, data transfer between the workers and SpatialOS seems to occur at the same rate regardless of layout. This will be due to the layout not having a significant effect on the number of cells at the boundary of the workers' domains. Alone, this evidence could point to the hexagonal and square grid layouts having equal performance and cell functionality processing could be the performance bottleneck.

A second hypothesis must also be true to show that this is the network data transfer limit. This hypothesis is that for a deployment operating below the assumed data transfer limit, the worker performance for a hexagonal layout deployment will be

improved. In nearly all instances, the average maximum worker load for the hexagonal deployment is around 20-25% lower. This is the expected effect if neighbour communication is the bottleneck as the hexagonal cell has 25% fewer neighbours than the square cell.

6.5 Client-side Prediction

As a result of the optimisation that reduced network updates for cells with neighbours all on the same worker, the client-side face validity of the simulation was compromised. The appearance indicated that the cells on the boundaries of the workers were the only ones involved in the simulation, except for every few seconds the other cells would also update. This was due to the client not receiving updates from the internal cells. To solve this, a client-side prediction extrapolation method was implemented which estimated the cells' concentrations if an update was not received, based on the two previous updates. This did not have any significant performance impact on the client (the average render time per deployment never differed from 16.66ms, 60fps). However, it also did not really serve any purpose that would advance understanding of using the platform for scientific experimentation. When creating games using a distributed system, an approach could be used whereby all workers and clients run deterministic prediction calculations. The network communications are used only to synchronise the predictions rather than to continuously share all data updates. However, when data accuracy is a priority this technique's use is limited.

6.6 Evaluation

The research question that this paper sought to answer concerns the identification of performance bottlenecks when deploying a simulation at scale. The developed application certainly identifies key areas where the design of simulation systems needs to be focussed to ameliorate such bottlenecks when targeting the SpatialOS platform. Whilst it was anticipated that the scale achievable would be far greater

before reaching the performance limitations, the aim was not to make the simulation more performant than using other techniques, only to identify what the limitations were of this specific technique and in this regard the results provided a good set of answers to the research question.

The hypothesis was that the load balancing capabilities of SpatialOS would mean that the number of workers could be increased in proportion to the scale of the simulation, to provide consistent performance at increasing scale. In fact, this was disproved for the simulation model implemented as the complexity of the model was found to be dependent on the number of workers as well as the scale of the simulation. Whilst the hypothesis was disproved for this implementation, as well as for a cursory implementation of an N-Body simulation, that does not mean it is unilaterally untrue. Instead it was revealed that a particular type of simulation would scale well on SpatialOS, however without major modifications the implemented models do not fit the profile of these types of simulation. An example of a simulation which would be better suited is the Manchester simulation discussed in the literature review. Here the simulation is comprised of autonomous agents which will perform various functionality and send commands to other agents when they desire to interact, this type of simulation has a much lower degree of worker-to-worker coupling.

The weaknesses of the simulation models implemented are partially down to the bias of the author toward simulations which are traditionally run using parallel GPGPU techniques, particularly simulations which benefit from a SIMD structure as discussed in the literature review. SpatialOS instead provides the opportunity to run MIMD structure applications at a previously inaccessible scale, as these types of simulation are badly optimised for GPGPU programming. Furthermore, the grid based Gray Scott simulation creates a high level of coupling between workers, even though it is a highly localised simulation as this locality persists across the borders of workers.

A significant area which was untested by this paper is the issue of synchronisation. The model chosen uses continuous data updates, rather than event-driven, which is less susceptible to desynchronization issues and the margin of error is therefore expected to be lower than in an event-driven simulation (though this is untested). However, when aiming for data accuracy even a simulation with continuous data updates would need to be highly synchronised and more research is required in this area to draw further conclusions. That said, for the reaction diffusion model, synchronisation would likely require further cross-worker communication and would therefore exacerbate the identified performance issues.

With hindsight, it would have been highly beneficial to use a much lower fixed framerate than the targeted 60fps. By lowering the framerate by as much as a factor of 10, the simulation would still be able to perform in real-time at a far greater scale as workers would have more time between frames to send and receive updates.

7. Conclusion

An application was constructed that modified the reaction diffusion simulation to run on the distributed platform SpatialOS. The application was functional across multiple workers in the cloud and provided the necessary complexity to be used for a performance analysis of the platform when being used for a real time accurate simulation.

The performance bottlenecks of the SpatialOS platform were identified for a complex simulation deployed at scale. For a simulation with a high amount of worker-to-worker coupling, network communication presents a significant performance barrier. Overcoming this barrier by using models with a lower degree of coupling or by creating novel techniques could unlock the huge scalability potential of cloud computing. The implementation of an alternate layout of the simulation shows that adaptations and model-specific modifications can be used to alleviate the restrictions imposed by the use of a distributed platform for hosting a simulation.

7.1 Further Work

The work presented shows that new methods of constructing simulation models need to be devised which make better use of the architectures available through cloud computing. In general, optimising a model further could be achieved by using general enhancements that could be applied to any distributed simulation (such as the neighbour communication optimisation implemented during this project) or more specific enhancements that depend on the exact model being implemented (such as the hexagonal grid implemented by this project). Some suggestions are presented which are based on the application delivered for this project.

The major bottleneck identified was network communications, therefore a way to optimise data transfer across the network needs to be found to introduce more scale into simulations such as these. One such method could be the introduction of a

'manager' using a Mediator design pattern (Gamma, 2015), this pattern's intent is to provide communicative objects with a common entity to communicate through. This could reduce the number of repeated data requests and provide a single point of communication across workers. This pattern was briefly investigated during development; however, it was decided that SpatialOS itself is a form of implementation of this pattern and alternative methods should be devised. With reflection a more granular mediator could be used on a per-worker basis which could improve data flow, although the distributed nature of SpatialOS and the lack of clearly defined domains for workers means that the exact implementation of this pattern will need a unique approach.

As briefly mentioned in the evaluation section, the reduction of framerate could provide a huge increase to the scale of simulation possible. It would be worthy of study to identify how far this will improve performance as it is dropped whilst still being within an acceptable range to provide real-time feedback and interactivity.

Another interesting point to study would be the use of prediction and interpolation methods using deterministic calculations to run the simulation. The frame-to-frame data accuracy is likely to be reduced, however the significant reduction in cross-worker communication would allow for far greater scale to be achievable. With increased scale comes the ability to observe different behaviours, even chaotic behaviour, that would be hard to observe at a smaller, more accurate scale.

8. References

Asanovic, K., Bodik, R., Catanzaro, B.C., Gebis, J.J., Husbands, P., Keutzer, K., Patterson, D.A., Plishker, W.L., Shalf, J., Williams, S.W. and Yelick, K.A. (2006) *The Landscape of Parallel Computing Research: A View from Berkeley.* Available at: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf. (Accessed: 20 April 2018).

Ball, P. (2009) *Shapes.* 1. publ. edn. Oxford [u.a.]: Oxford Univ. Press.

Barrett, C., Bisset, K., Eubank, S., Feng, X. and Marathe, M. (Nov 15, 2008) *EpiSimdemics: An efficient algorithm for simulating the spread of infectious disease over large realistic social networks.* IEEE Press, pp. 1.

Braddock, R., Claunch, M. and Rainbolt, J. (Mar 1, 1992) *Operational performance metrics in a distributed system. Part II.* ACM, pp. 873.

CCP. (2013) *EVE Online* [Video Game]. CCP Available at: https://www.eveonline.com/. (Downloaded: February 2011).

Dematté, L. and Prandi, D. (2010) 'GPU computing for systems biology', *Briefings in bioinformatics,* 11(3), pp. 323-333. doi: 10.1093/bib/bbq006.

Dirksen, R. (2013) *Synchronization in a Distributed System.* Available at: https://8thlight.com/blog/rylan-dirksen/2013/10/04/synchronization-in-a-distributed-system.html (Accessed: 03 February 2018).

Drain, B. (2008) *EVE Evolved: EVE Online's server model.* Available at: https://www.engadget.com/2008/09/28/eve-evolved-eve-onlines-server-model/ (Accessed: 03 April 2018).

Franklin-Wallis, O. (2017) *If we're living in a simulation, this UK startup probably built it.* Available at: http://www.wired.co.uk/article/improbable-quest-to-build-the-matrix (Accessed: 08 December 2017).

Fujimoto, R.M. (2003) *Distributed simulation systems.* 2003. pp. 124.

Gamma, E. (2015) *Design patterns.* 2. ed. edn. Boston [u.a.]: Addison-Wesley.

Hut, P. and Barnes, J. (1986) 'A hierarchical O(N log N) force-calculation algorithm', *Nature,* 324(6096), pp. 446-449. doi: 10.1038/324446a0.

Improbable (2016) *What we found when we simulated the backbone of the entire Internet on SpatialOS.* Available at: https://improbable.io/company/news/2016/03/24/what-we-found-when-we-simulated-the-backbone-of-the-entire-internet-on-spatialos (Accessed: 04 December 2017).

Lees, M., Logan, B. and Theodoropoulos, G. (June, 2003) *Adaptive optimistic synchronisation for multi-agent distributed simulation*. 2003.

Parkin, S. (2016) *Highly Detailed City Simulation Is the New Autonomous Taxi Dispatch.* Available at: https://www.technologyreview.com/s/601663/highly-detailed-city-simulation-is-the-new-autonomous-taxi-dispatch/ (Accessed: 04 December 2017).

Phillips, J.C., Gengbin Zheng, Kumar, S. and Kale, L.V. (2002) *NAMD: Biomolecular Simulation on Thousands of Processors.* IEEE, pp. 36.

Richmond, P., Walker, D., Coakley, S. and Romano, D. (2010) 'High performance cellular level agent-based simulation with FLAME for the GPU', *Briefings in bioinformatics,* 11(3), pp. 334-347. doi: 10.1093/bib/bbp073.

Sims, K. (2013) *Reaction-Diffusion Tutorial.* Available at: http://www.karlsims.com/rd.html (Accessed: 04 February 2018).

Stefanescu, T. (2006) *EVE Online Readies the Largest Supercomputer in the Gaming Industry*. Available at: https://news.softpedia.com/news/EVE-Online-Readies-the-Largest-Supercomputer-in-the-Gaming-Industry-35225.shtml (Accessed: 06 April 2018).

Sunar, S. (2016) *Improbable Joins Forces with Google to Empower Developers to Build New Online Gaming Experiences With Strategic Partnership & Open Game Developer Alpha*. Available at: https://www.businesswire.com/news/home/20161213006089/en/Improbable-Joins-Forces-Google-Empower-Developers-Build (Accessed: 04 December 2017).

Vigelius, M., Lane, A. and Meyer, B. (2011) 'Accelerating reaction-diffusion simulations with general-purpose graphics processing units', *Bioinformatics (Oxford, England)*, 27(2), pp. 288-290. doi: 10.1093/bioinformatics/btq622.

9. Bibliography

Flexible Large Scale Agent Modelling Environment for the GPU (FLAMEGPU). Available at: http://www.flamegpu.com/home(Accessed: September 20, 2017).

Collier, N. and North, M. (2012) 'Repast HPC: A Platform for Large-Scale Agent-Based Modeling', in Dubitzky, W., Kurowski, K. and Schott, B. (eds.) *Large-Scale Computing* Hoboken, NJ, USA: John Wiley & Sons, Inc, pp. 81-109.

Cuomo, A., Rak, M. and Villano, U. (2015) 'Performance Prediction of Cloud Applications Through Benchmarking and Simulation', *International Journal of Computational Science and Engineering,* 11(1), pp. 46. doi: 10.1504/IJCSE.2015.071362.

Eriksson, H., Raciti, M., Basile, M., Cunsolo, A., Fröberg, A., Leifler, O., Ekberg, J. and Timpka, T. (2011) 'A Cloud-Based Simulation Architecture for Pandemic Influenza Simulation', *AMIA ... Annual Symposium proceedings / AMIA Symposium. AMIA Symposium,* 2011, pp. 364.

Improbable @ GDC '17 | a look inside SpatialOS (2017) Game Developer's Conference.

Himanshu Kaul, Zhanfeng Cui and Yiannis Ventikos (2013) 'A Multi-Paradigm Modeling Framework to Simulate Dynamic Reciprocity in a Bioreactor', *PLoS One,* 8(3). doi: 10.1371/journal.pone.0059671.

Mikkonen, T. (Apr 1, 1998) *Formalizing design patterns.* IEEE Computer Society, pp. 115.

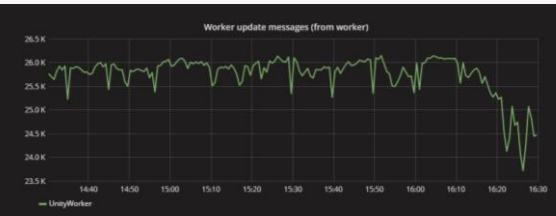
Niazi, M. and Hussain, A. (2011) 'Agent-based computing from multi-agent systems to agent-based models: a visual survey', *Scientometrics,* 89(2), pp. 479-499. doi: 10.1007/s11192-011-0468-9.

Resat, H., Costa, M.N. and Shankaran, H. (2011) 'Spatial Aspects in Biological System Simulations' *Methods in Enzymology* United States: Elsevier Science & Technology, pp. 485-511.

10. Appendix

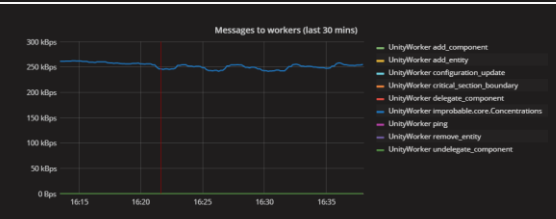
10.1 441 cells simulated using the base functionality

10.1.1 441 cells simulated across 4 workers

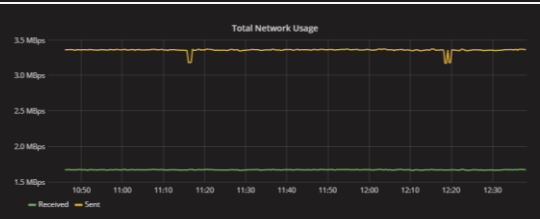


The rate of messages sent from an average worker to the SpatialOS platform per second.

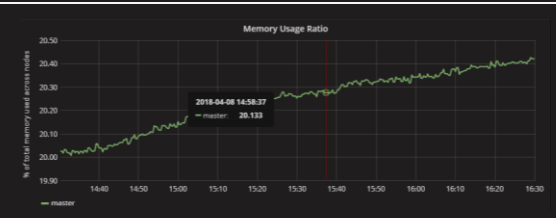
The rate of messages sent from SpatialOS to the average worker per second.



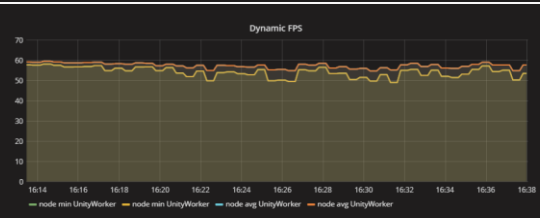
The message composition of messages sent to an average worker per second.



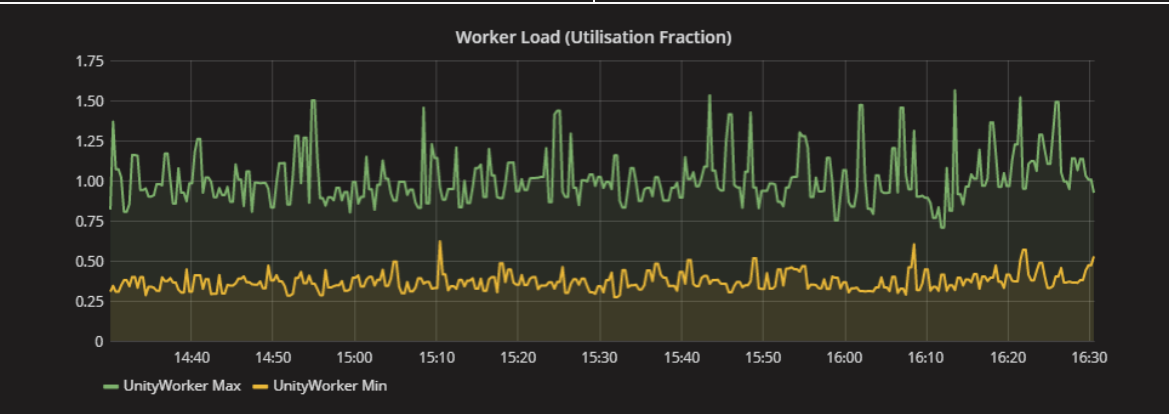
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



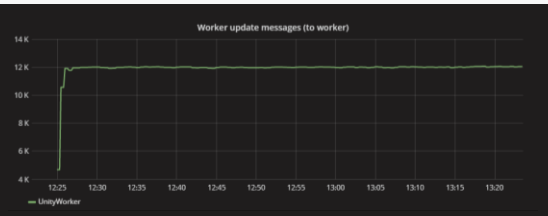
The average framerate of a worker.



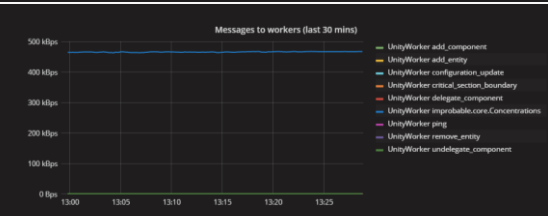
The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.1.2 441 cells simulated across 9 workers

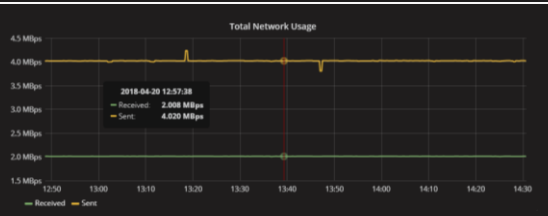
The rate of messages sent from an average worker to the SpatialOS platform per second.



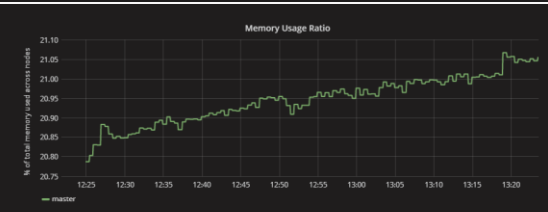
The rate of messages sent from SpatialOS to the average worker per second.



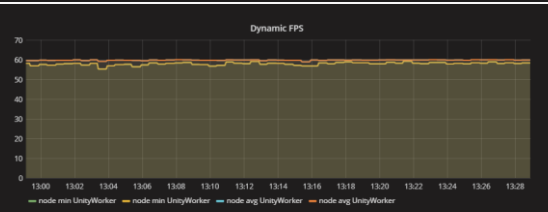
The message composition of messages sent to an average worker per second.



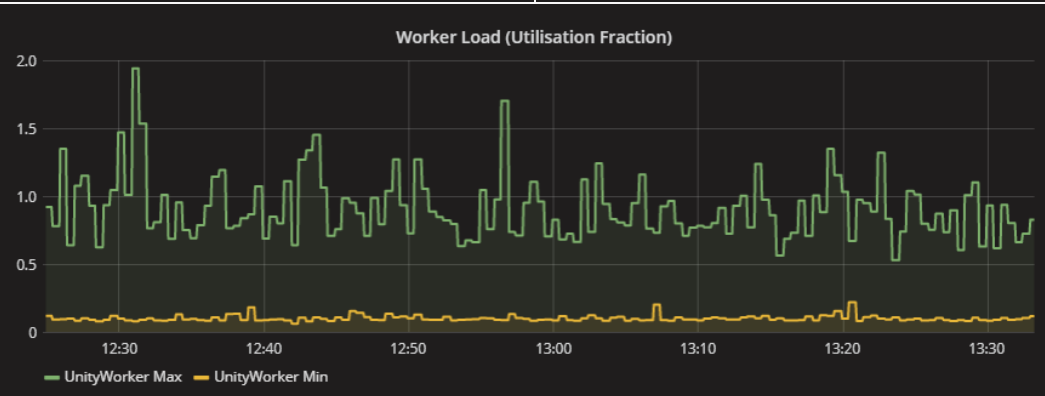
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



The average framerate of a worker.



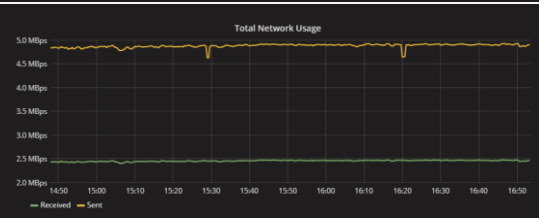
The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.1.3 441 cells simulated across 16 workers

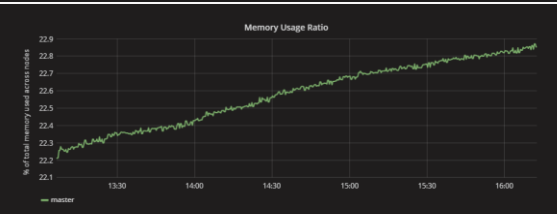
The rate of messages sent from an average worker to the SpatialOS platform per second.

The rate of messages sent from SpatialOS to the average worker per second.

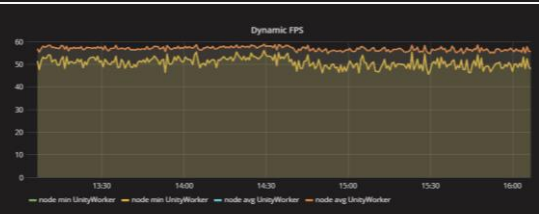
The message composition of messages sent to an average worker per second.



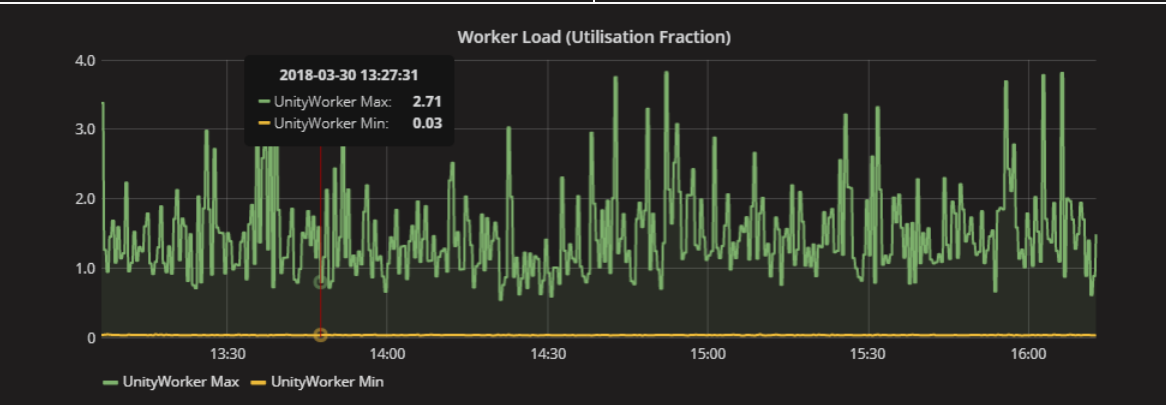
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



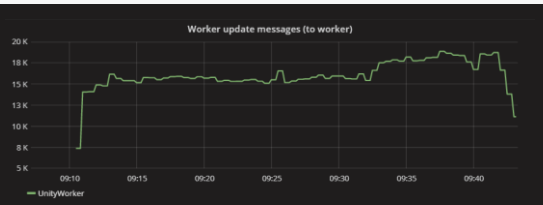
The average framerate of a worker.



The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

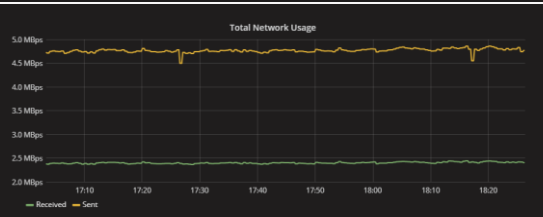
10.1.4 441 cells simulated across 25 workers

The rate of messages sent from an average worker to the SpatialOS platform per second.

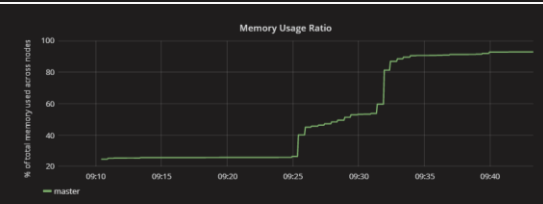


The rate of messages sent from SpatialOS to the average worker per second.

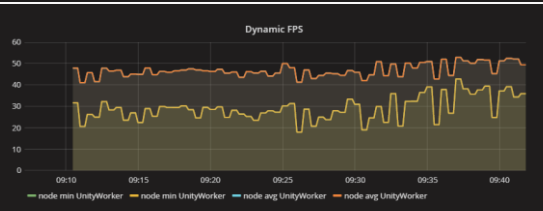
The message composition of messages sent to an average worker per second.



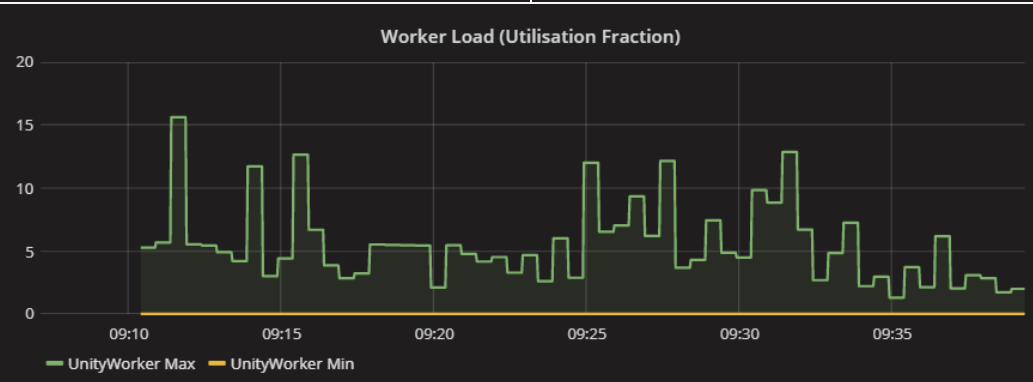
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



The average framerate of a worker.



The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

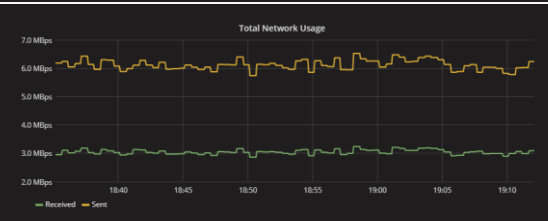
10.2 441 cells simulated using base functionality and random worker placement

10.2.1 441 cells simulated across 25 workers

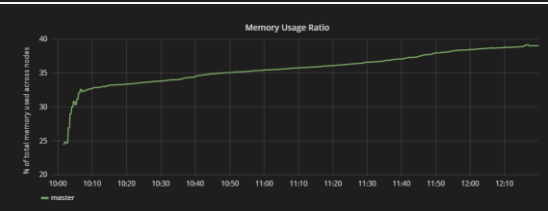
The rate of messages sent from an average worker to the SpatialOS platform per second.

The rate of messages sent from SpatialOS to the average worker per second.

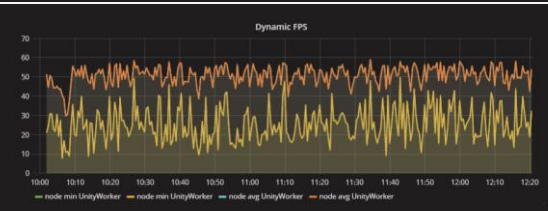
The message composition of messages sent to an average worker per second.



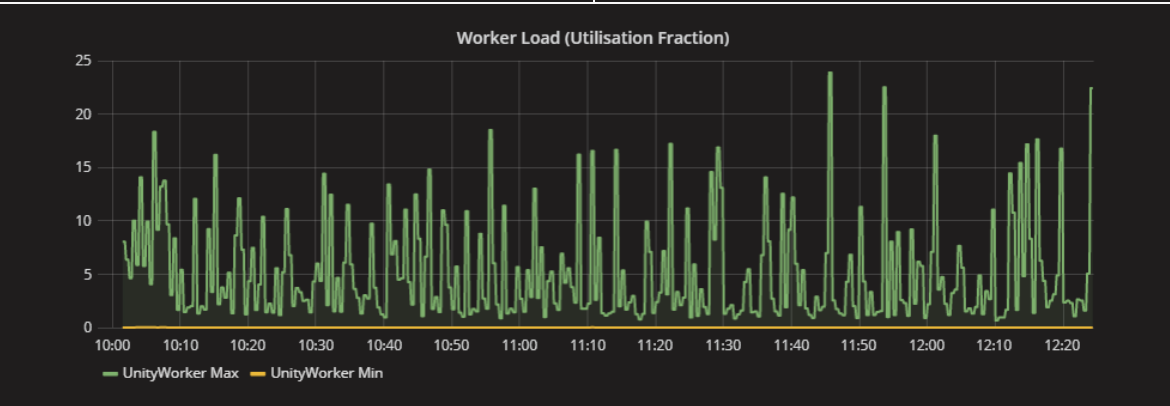
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



The average framerate of a worker.



The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

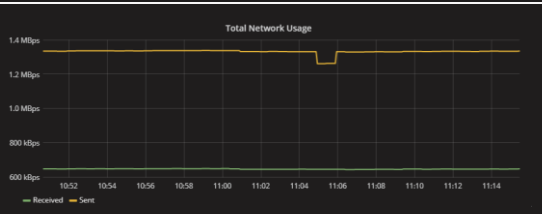
10.3 441 cells simulated using a communication frequency optimisation

10.3.1 441 cells simulated across 4 workers

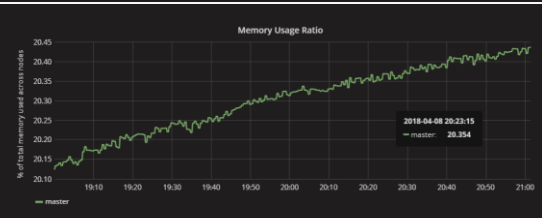
The rate of messages sent from an average worker to the SpatialOS platform per second.

The rate of messages sent from SpatialOS to the average worker per second.

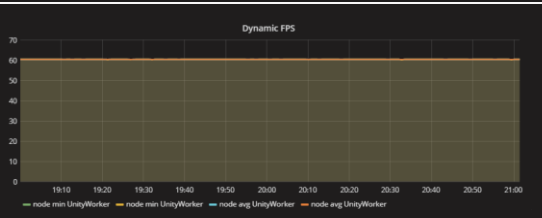
The message composition of messages sent to an average worker per second.



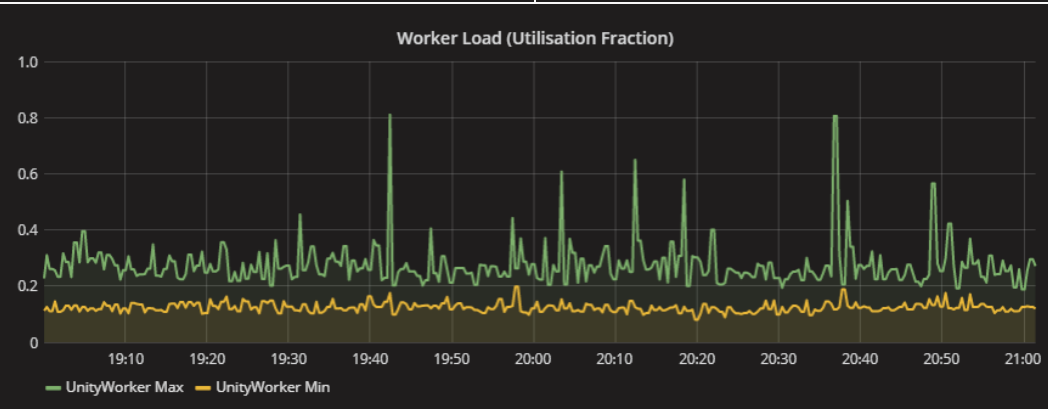
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.

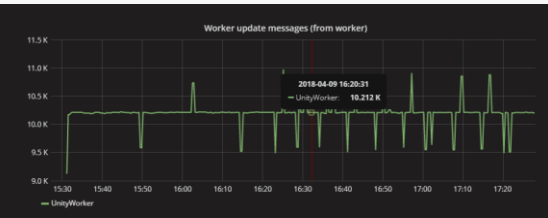


The average framerate of a worker.



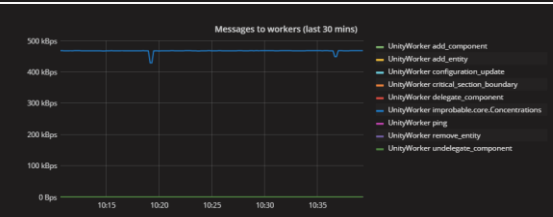
The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.3.2 441 cells simulated across 9 workers

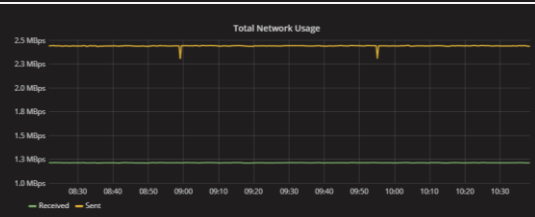


The rate of messages sent from an average worker to the SpatialOS platform per second.

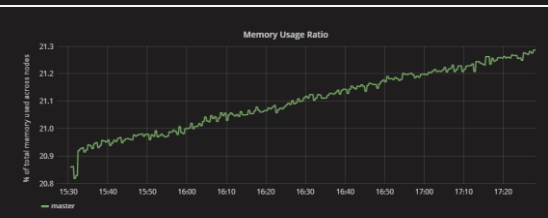
The rate of messages sent from SpatialOS to the average worker per second.



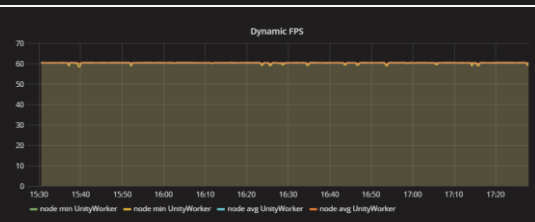
The message composition of messages sent to an average worker per second.



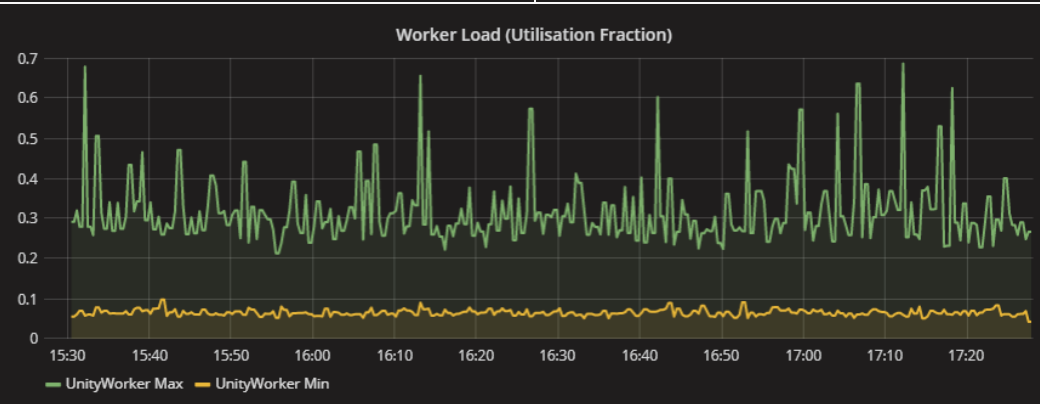
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



The average framerate of a worker.



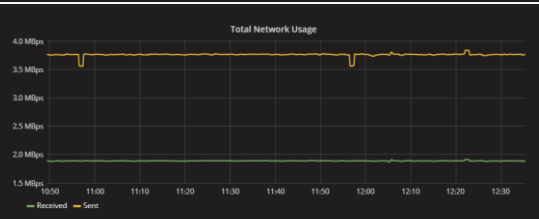
The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.3.3 441 cells simulated across 16 workers

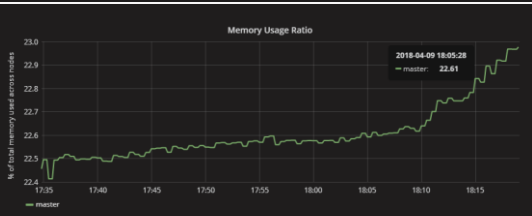
The rate of messages sent from an average worker to the SpatialOS platform per second.

The rate of messages sent from SpatialOS to the average worker per second.

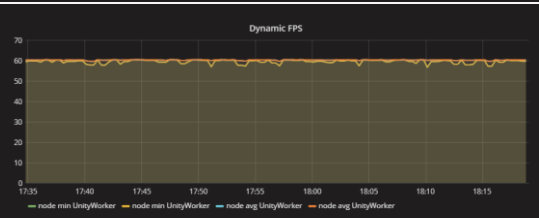
The message composition of messages sent to an average worker per second.



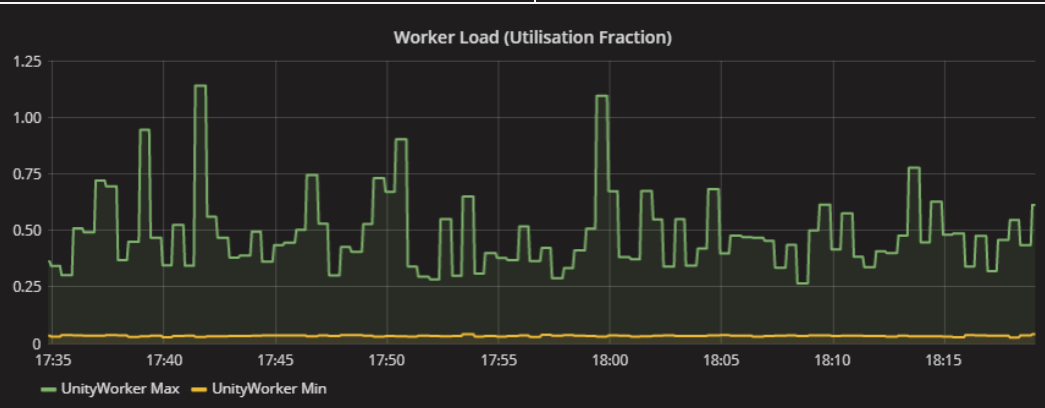
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



The average framerate of a worker.



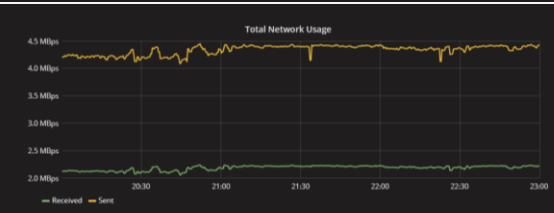
The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.3.4 441 cells simulated across 25 workers

The rate of messages sent from an average worker to the SpatialOS platform per second.

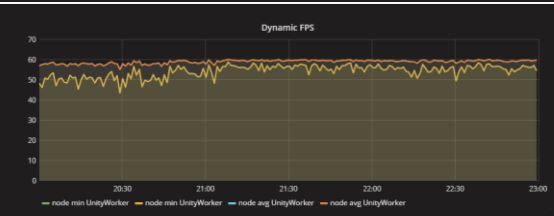
The rate of messages sent from SpatialOS to the average worker per second.

The message composition of messages sent to an average worker per second.

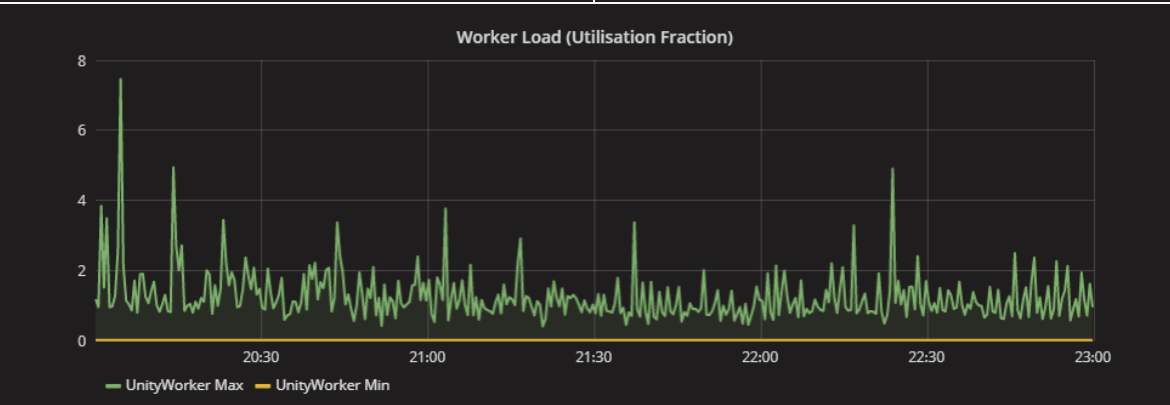


The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.

The amount of memory used as a percentage of the total available memory available across the system.



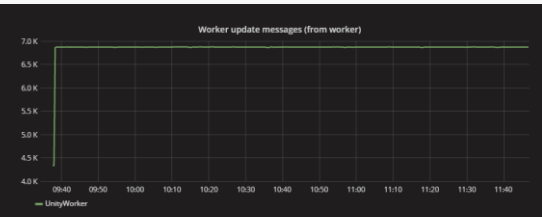
The average framerate of a worker.



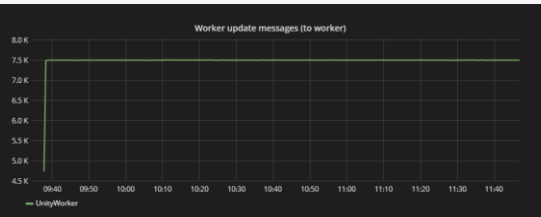
The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.4 625 cells simulated using a communication frequency optimisation

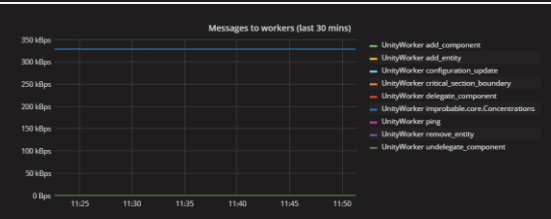
10.4.1 625 cells simulated across 4 workers



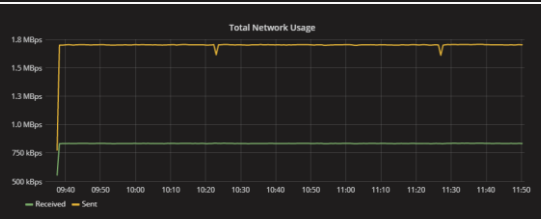
The rate of messages sent from an average worker to the SpatialOS platform per second.



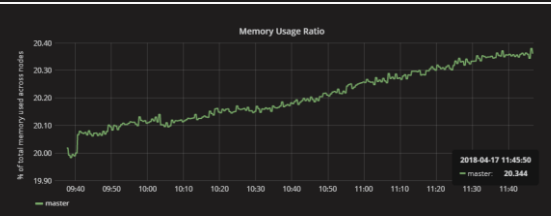
The rate of messages sent from SpatialOS to the average worker per second.



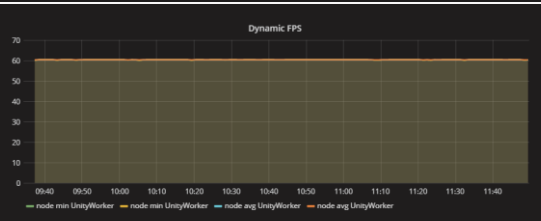
The message composition of messages sent to an average worker per second.



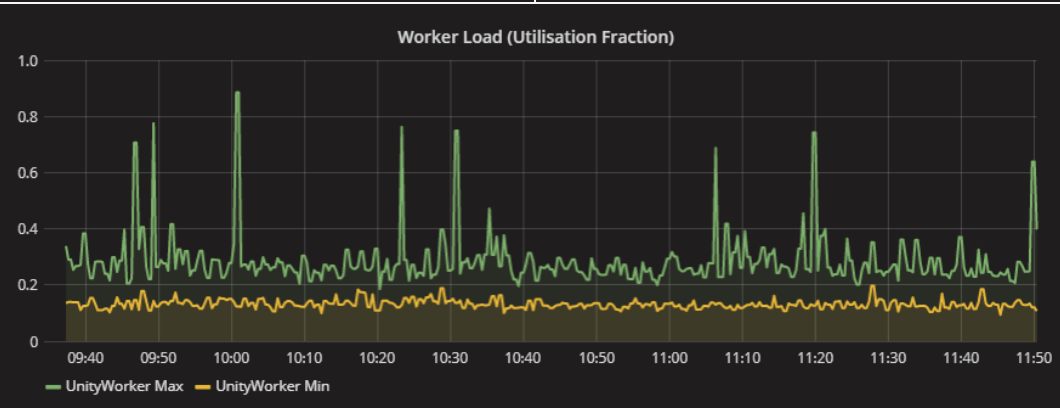
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.

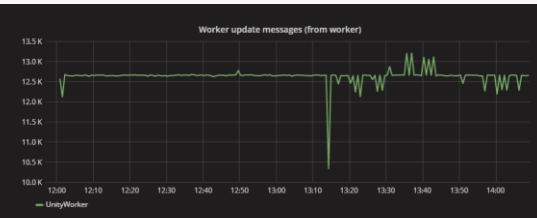


The average framerate of a worker.



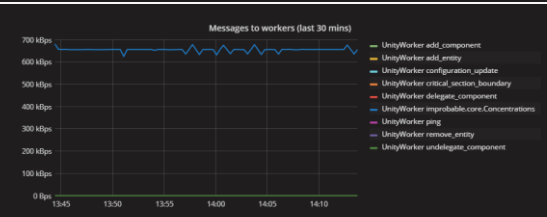
The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.4.2 625 cells simulated across 9 workers

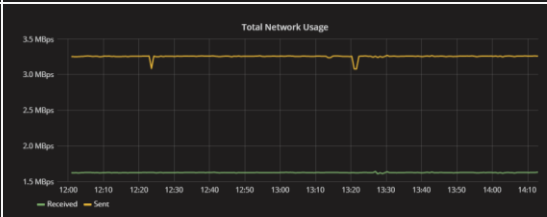


The rate of messages sent from an average worker to the SpatialOS platform per second.

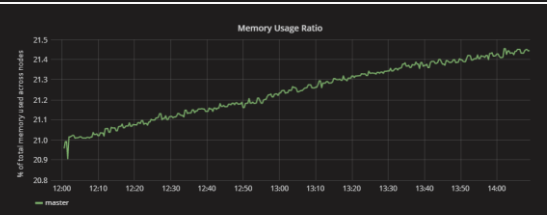
The rate of messages sent from SpatialOS to the average worker per second.



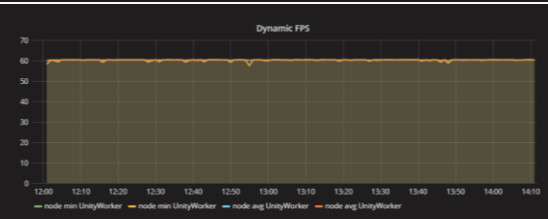
The message composition of messages sent to an average worker per second.



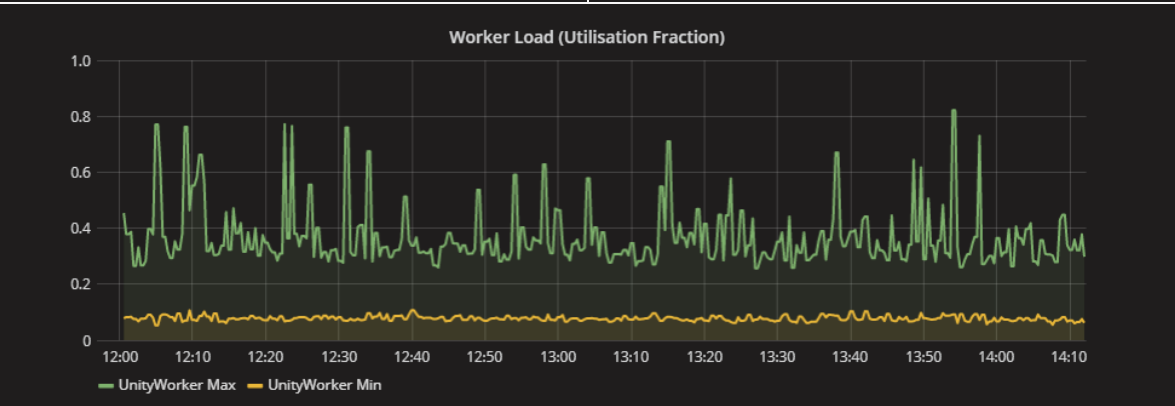
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



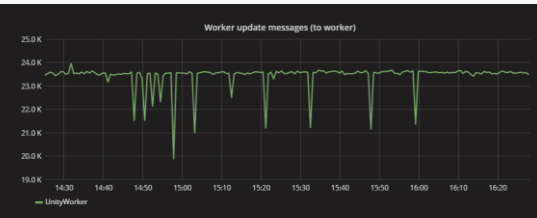
The average framerate of a worker.



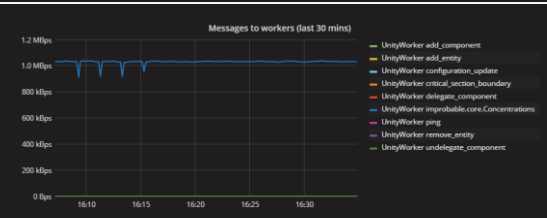
The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.4.3 625 cells simulated across 16 workers

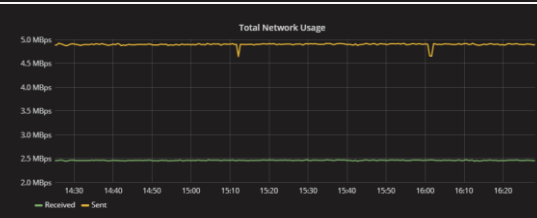
The rate of messages sent from an average worker to the SpatialOS platform per second.



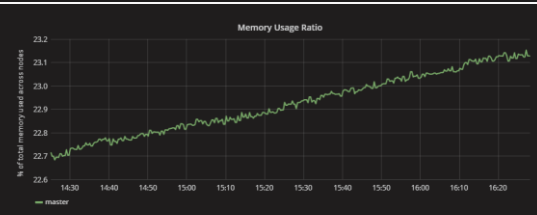
The rate of messages sent from SpatialOS to the average worker per second.



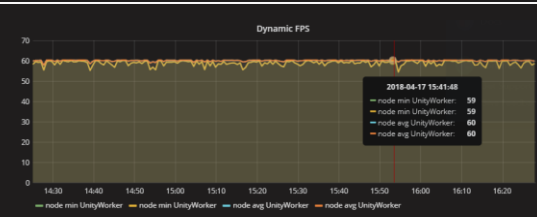
The message composition of messages sent to an average worker per second.



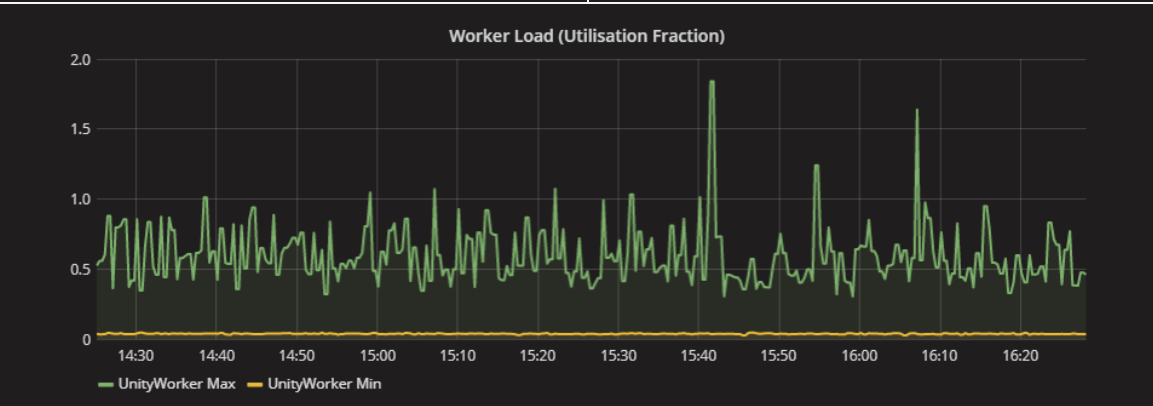
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



The average framerate of a worker.

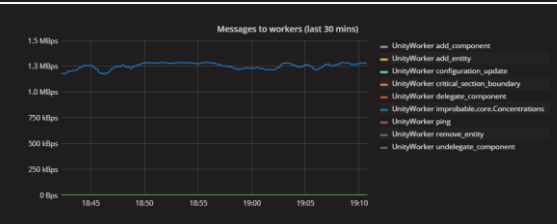


The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

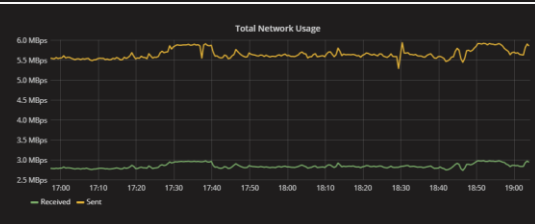
10.4.4 625 cells simulated across 25 workers

The rate of messages sent from an average worker to the SpatialOS platform per second.

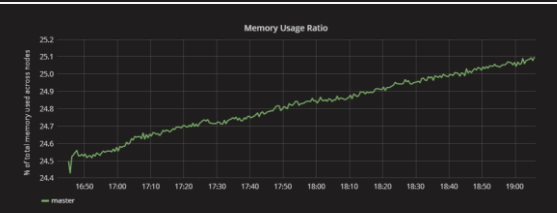
The rate of messages sent from SpatialOS to the average worker per second.



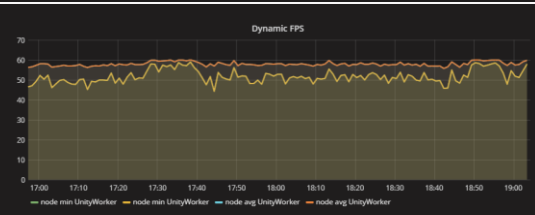
The message composition of messages sent to an average worker per second.



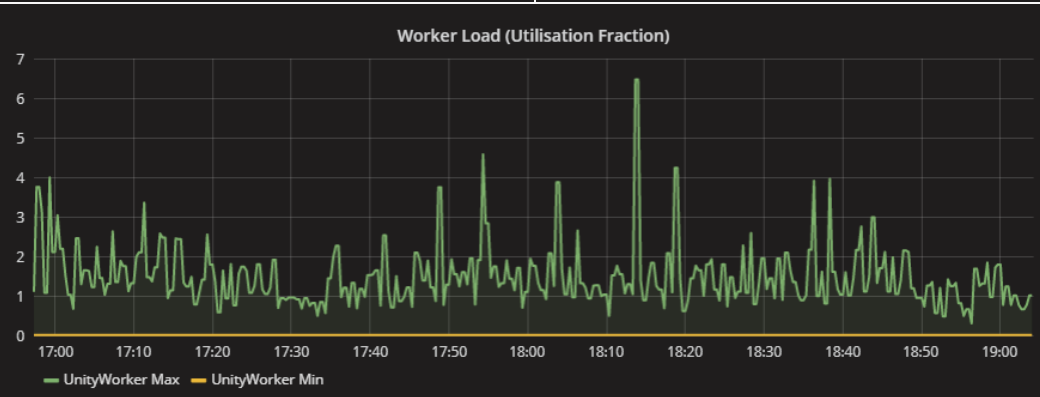
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



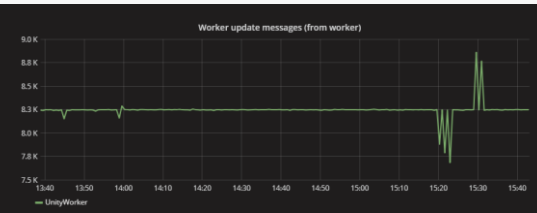
The average framerate of a worker.



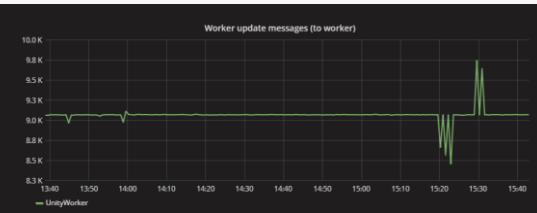
The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.5 900 cells simulated using a communication frequency optimisation

10.5.1 900 cells simulated across 4 workers

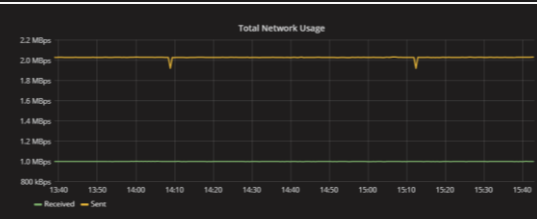


The rate of messages sent from an average worker to the SpatialOS platform per second.

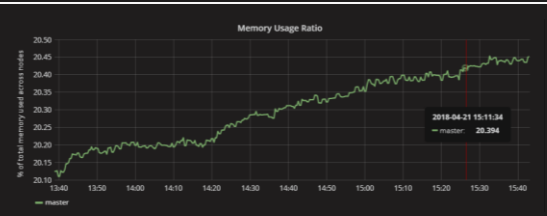


The rate of messages sent from SpatialOS to the average worker per second.

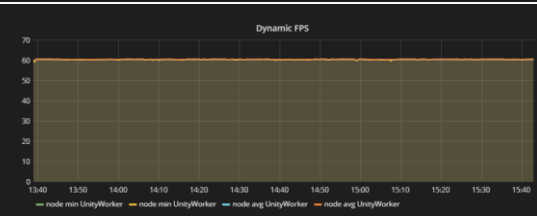
The message composition of messages sent to an average worker per second.



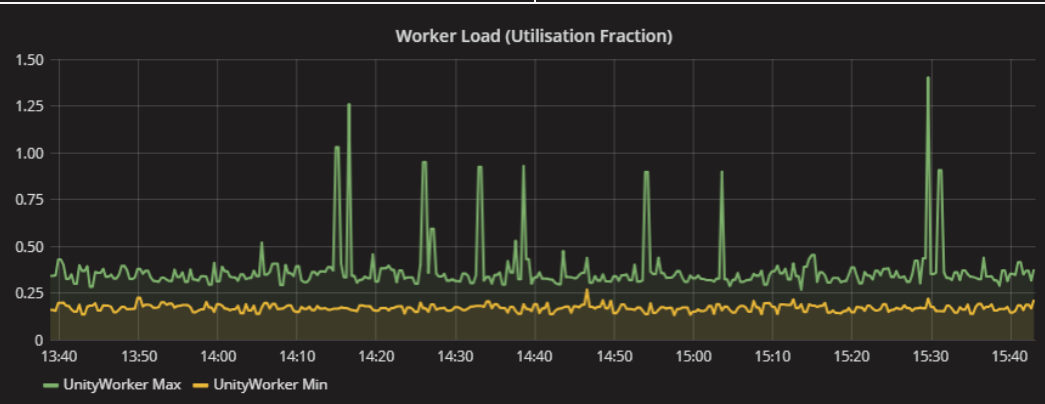
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



The average framerate of a worker.



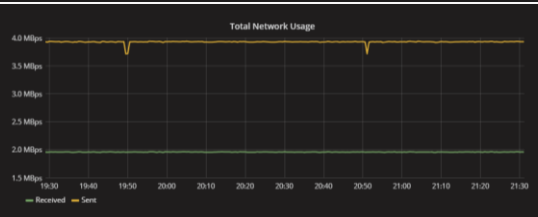
The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.5.2 900 cells simulated across 9 workers

The rate of messages sent from an average worker to the SpatialOS platform per second.

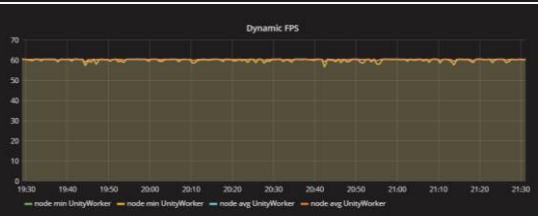
The rate of messages sent from SpatialOS to the average worker per second.

The message composition of messages sent to an average worker per second.

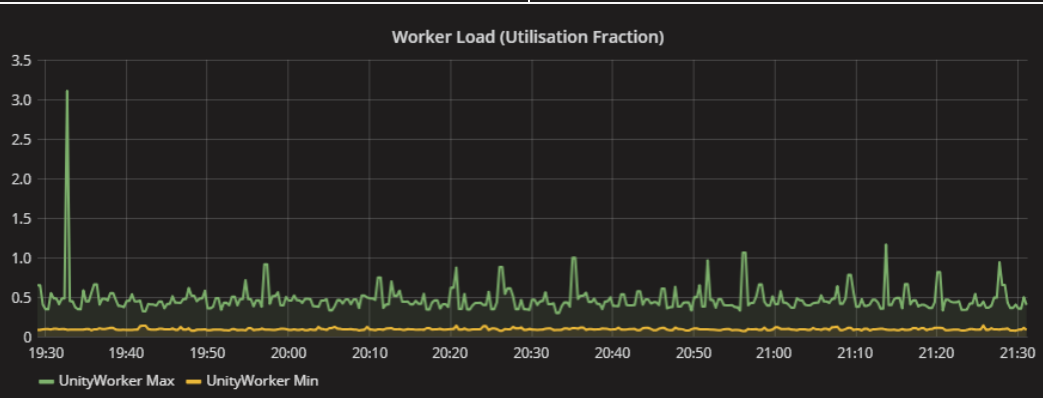


The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.

The amount of memory used as a percentage of the total available memory available across the system.



The average framerate of a worker.

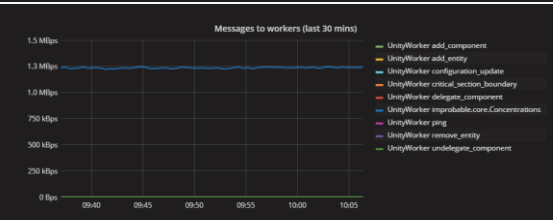


The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

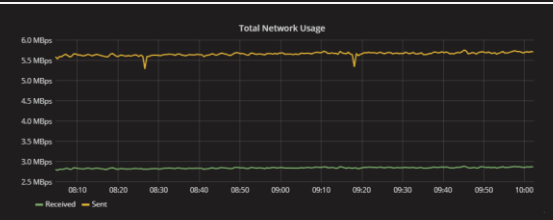
10.5.3 900 cells simulated across 16 workers

The rate of messages sent from an average worker to the SpatialOS platform per second.

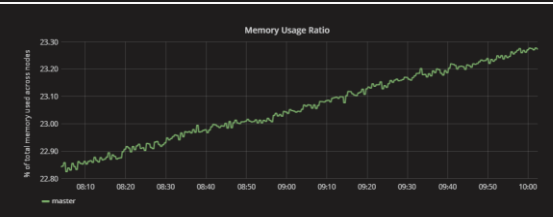
The rate of messages sent from SpatialOS to the average worker per second.



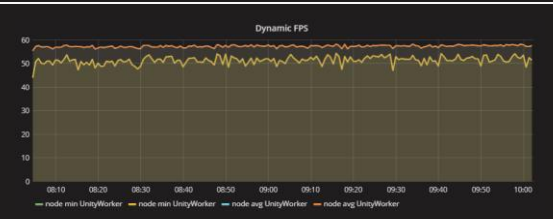
The message composition of messages sent to an average worker per second.



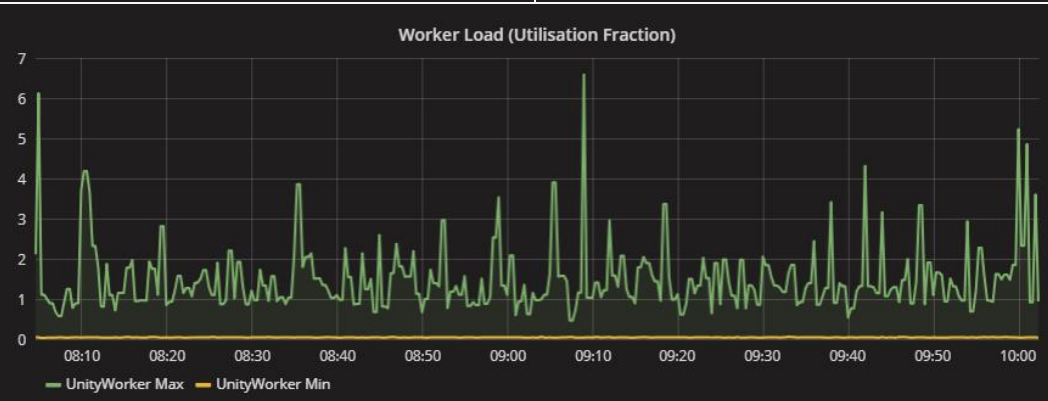
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.

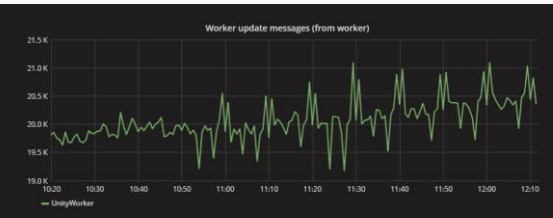


The average framerate of a worker.

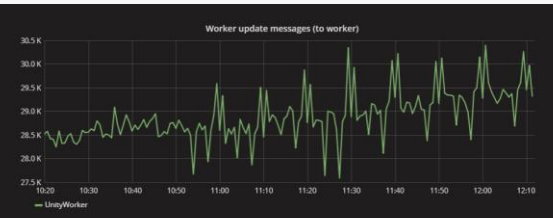


The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

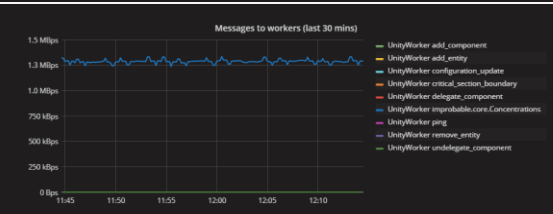
10.5.4 900 cells simulated across 25 workers



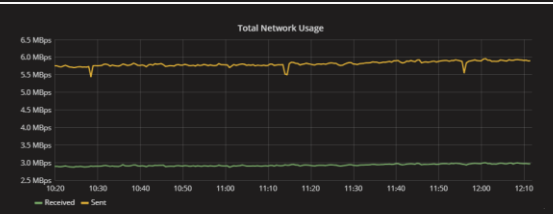
The rate of messages sent from an average worker to the SpatialOS platform per second.



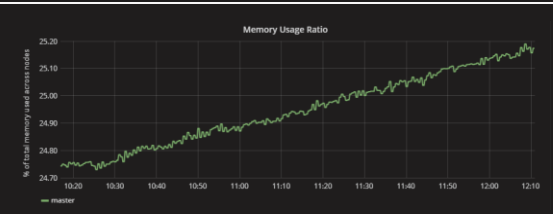
The rate of messages sent from SpatialOS to the average worker per second.



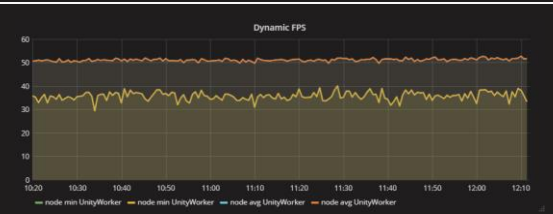
The message composition of messages sent to an average worker per second.



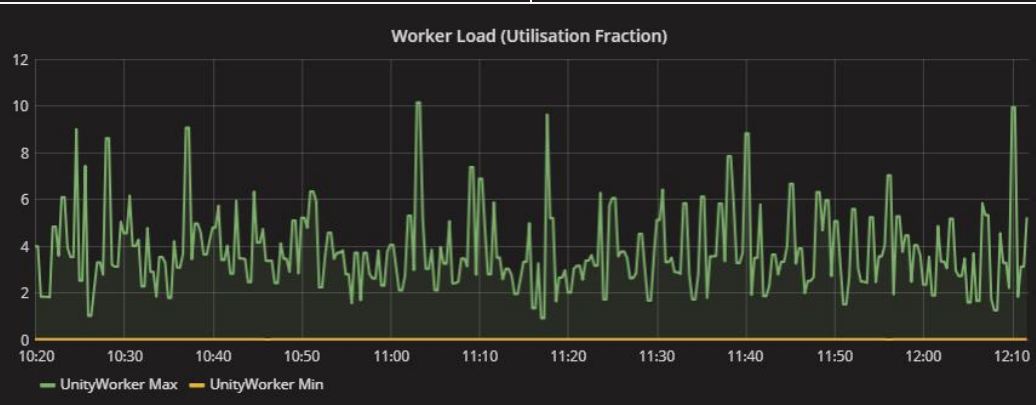
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



The average framerate of a worker.



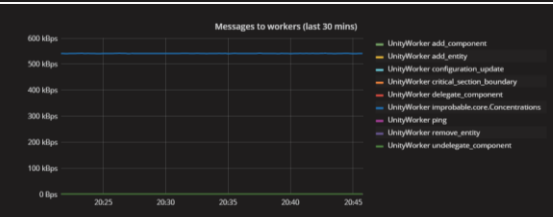
The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.6 1600 cells simulated using a communication frequency optimisation

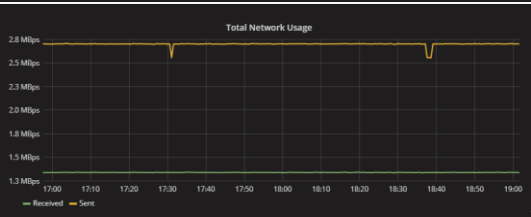
10.6.1 1600 cells simulated across 4 workers

The rate of messages sent from an average worker to the SpatialOS platform per second.

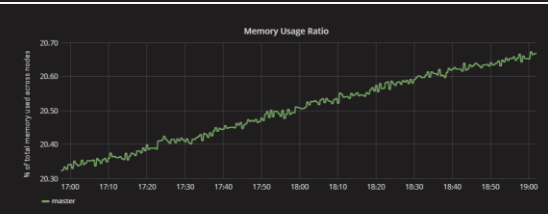
The rate of messages sent from SpatialOS to the average worker per second.



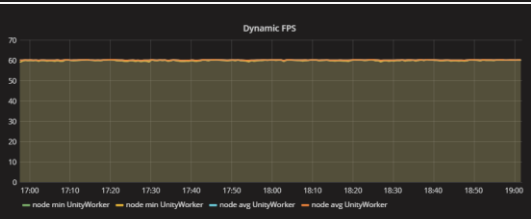
The message composition of messages sent to an average worker per second.



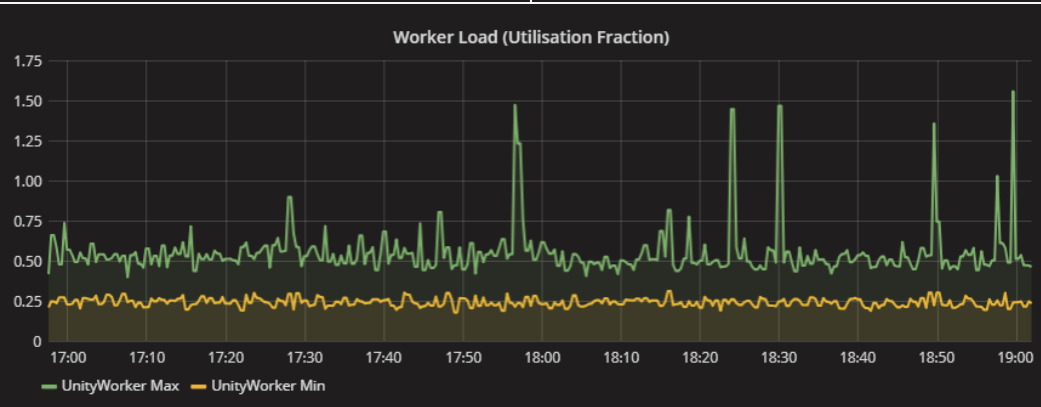
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



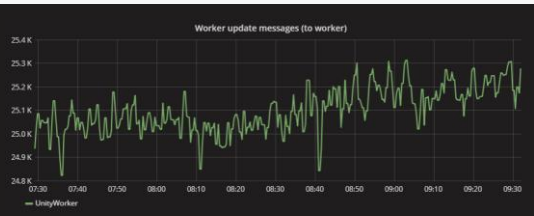
The average framerate of a worker.



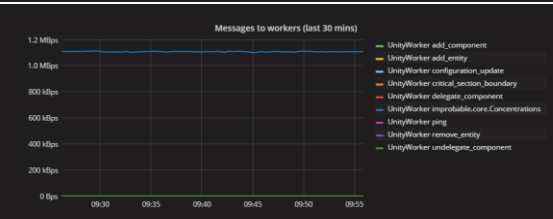
The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.6.2 1600 cells simulated across 9 workers

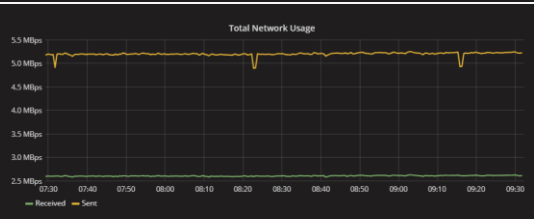
The rate of messages sent from an average worker to the SpatialOS platform per second.



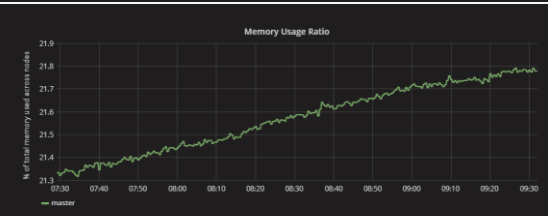
The rate of messages sent from SpatialOS to the average worker per second.



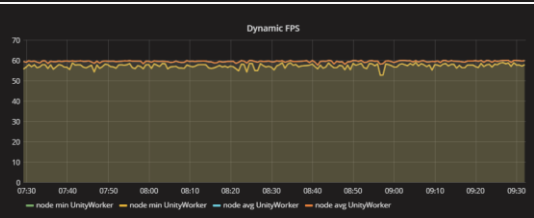
The message composition of messages sent to an average worker per second.



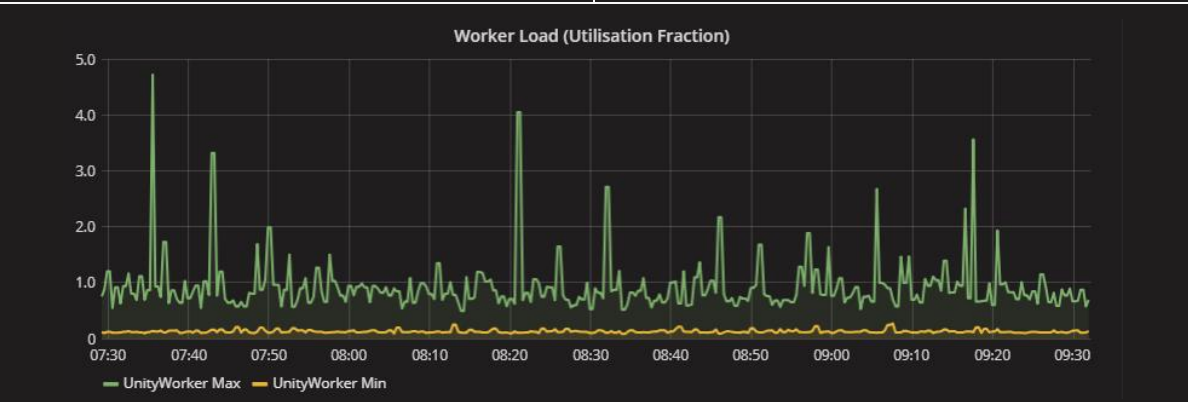
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



The average framerate of a worker.



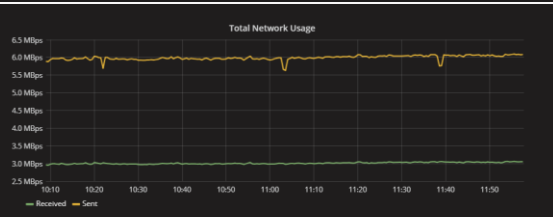
The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.6.3 1600 cells simulated across 16 workers

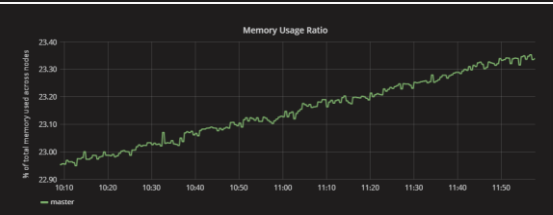
The rate of messages sent from an average worker to the SpatialOS platform per second.

The rate of messages sent from SpatialOS to the average worker per second.

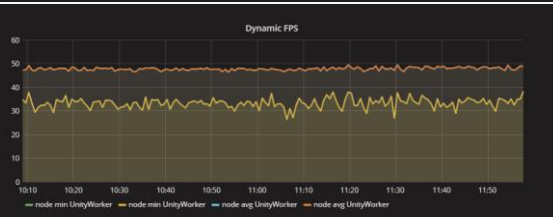
The message composition of messages sent to an average worker per second.



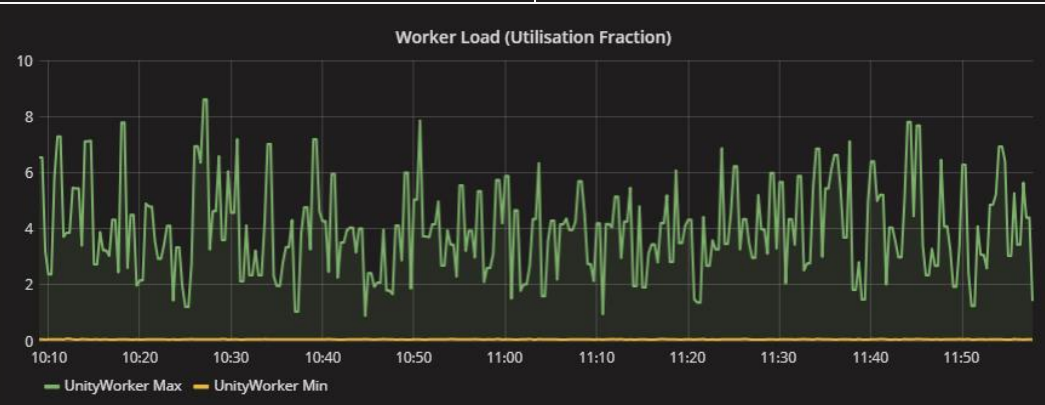
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



The average framerate of a worker.

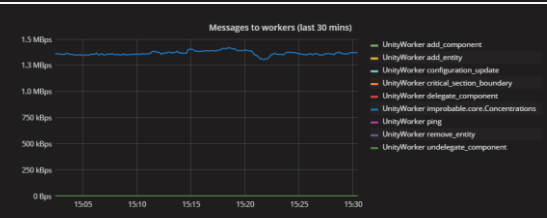


The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

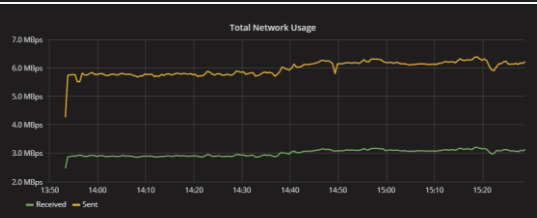
10.6.4 1600 cells simulated across 25 workers

The rate of messages sent from an average worker to the SpatialOS platform per second.

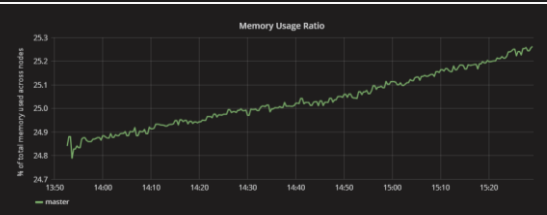
The rate of messages sent from SpatialOS to the average worker per second.



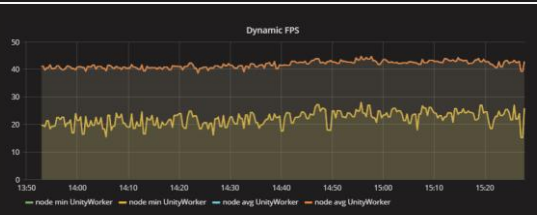
The message composition of messages sent to an average worker per second.



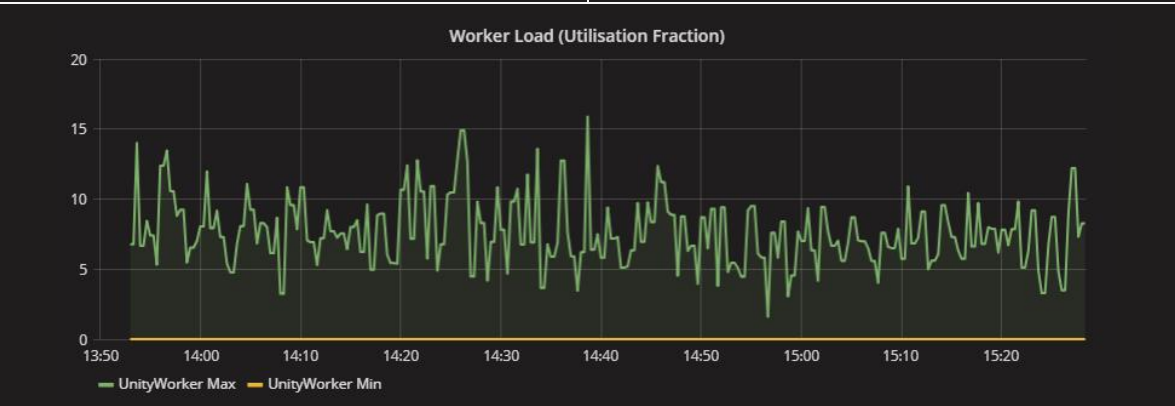
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



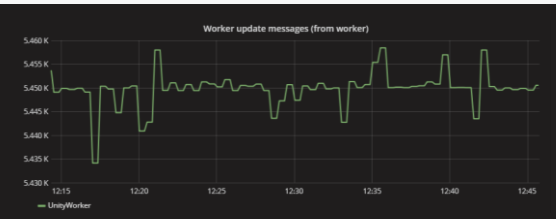
The average framerate of a worker.



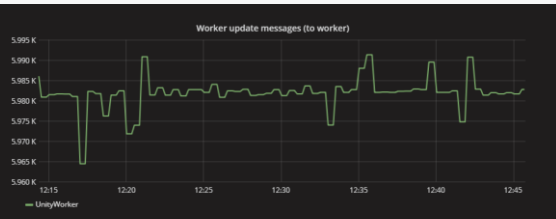
The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.7 441 cells simulated using a communication optimisation and a hexagonal layout

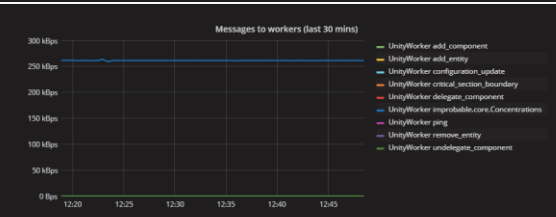
10.7.1 441 cells simulated across 4 workers



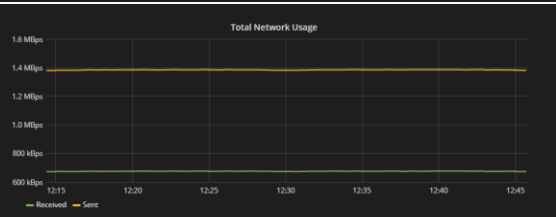
The rate of messages sent from an average worker to the SpatialOS platform per second.



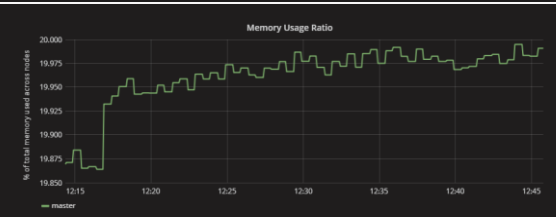
The rate of messages sent from SpatialOS to the average worker per second.



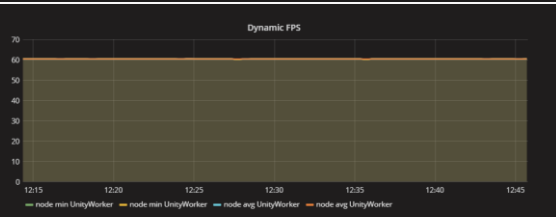
The message composition of messages sent to an average worker per second.



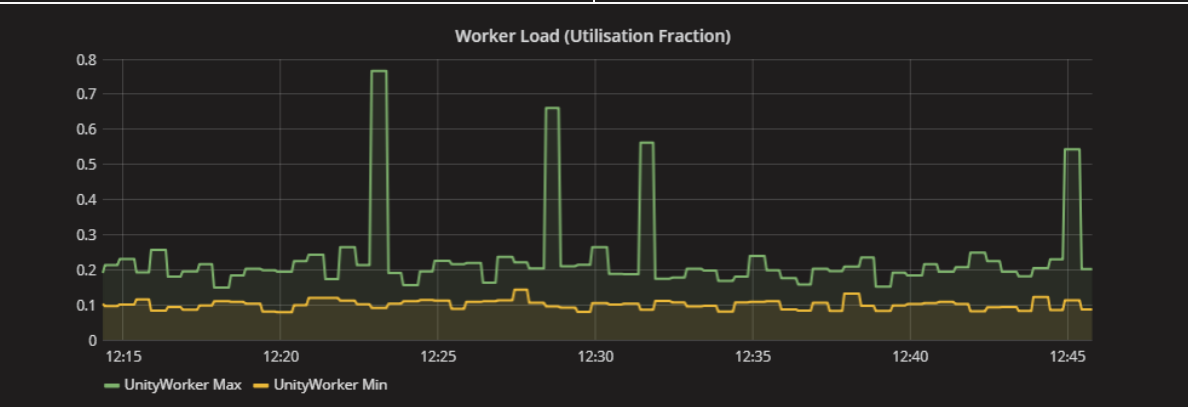
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



The average framerate of a worker.



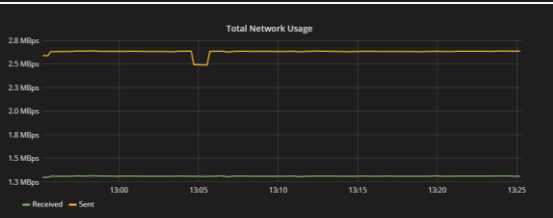
The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.7.2 441 cells simulated across 9 workers

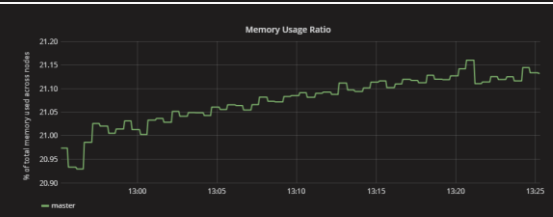
The rate of messages sent from an average worker to the SpatialOS platform per second.

The rate of messages sent from SpatialOS to the average worker per second.

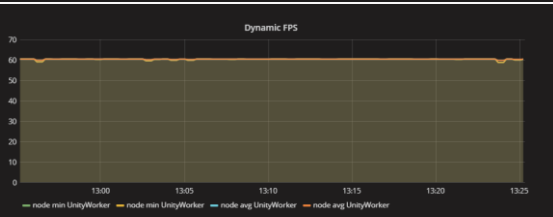
The message composition of messages sent to an average worker per second.



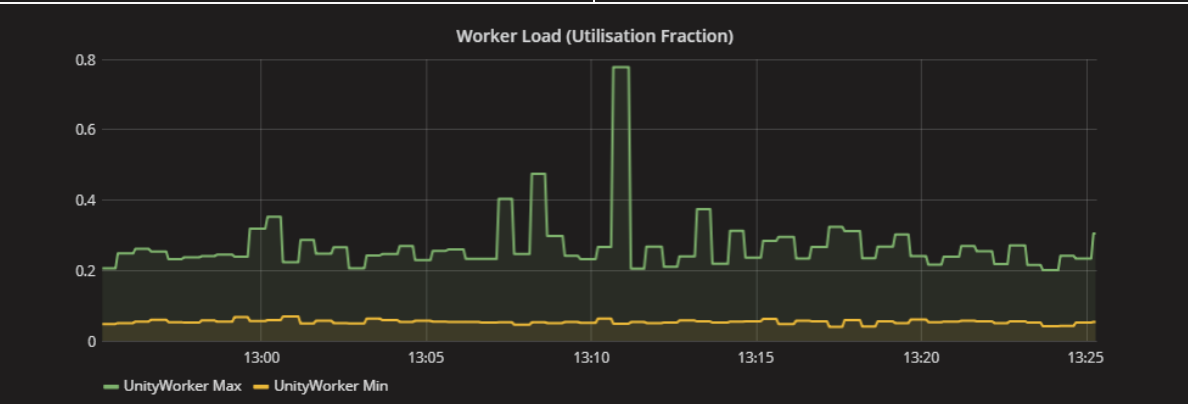
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



The average framerate of a worker.



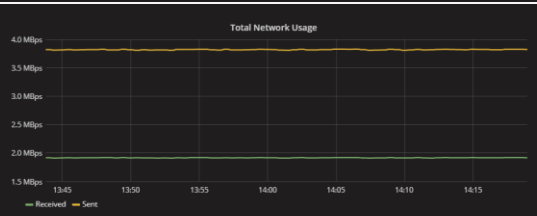
The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.7.3 441 cells simulated across 16 workers

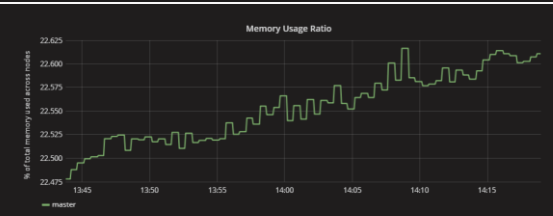
The rate of messages sent from an average worker to the SpatialOS platform per second.

The rate of messages sent from SpatialOS to the average worker per second.

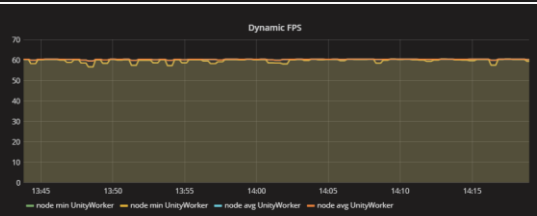
The message composition of messages sent to SpatialOS from an average worker per second.



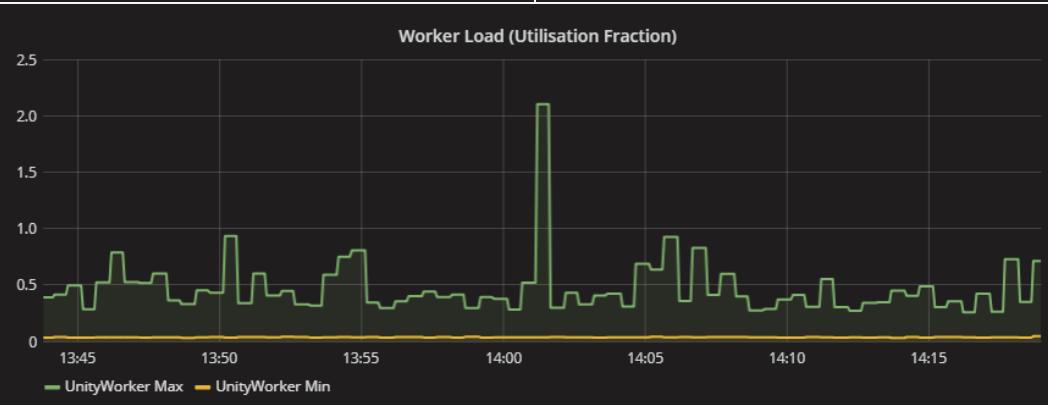
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



The average framerate of a worker.



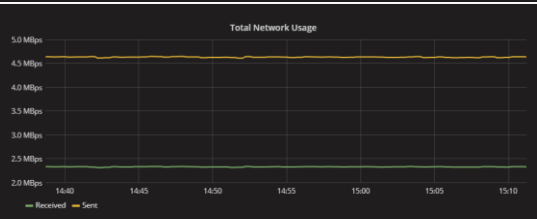
The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.7.4 441 cells simulated across 25 workers

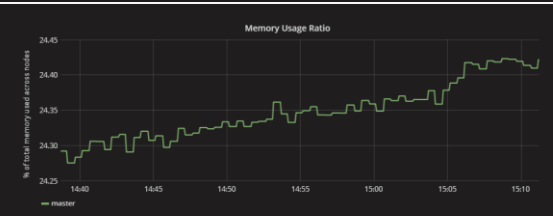
The rate of messages sent from an average worker to the SpatialOS platform per second.

The rate of messages sent from SpatialOS to the average worker per second.

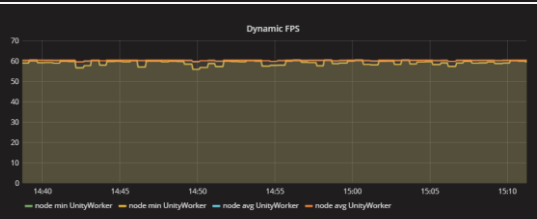
The message composition of messages sent to SpatialOS from an average worker per second.



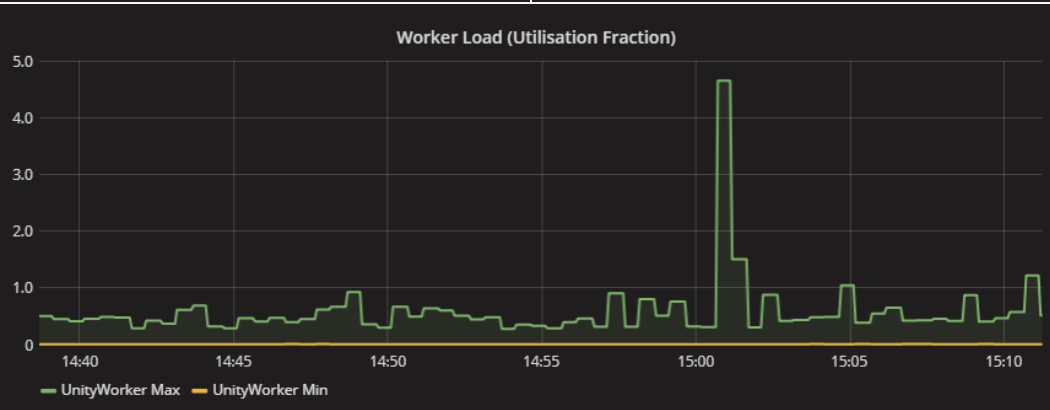
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



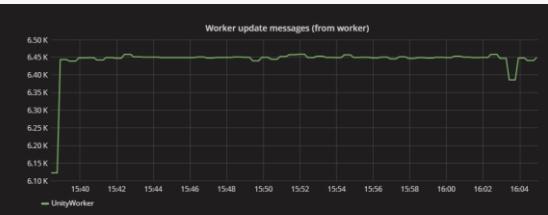
The average framerate of a worker.



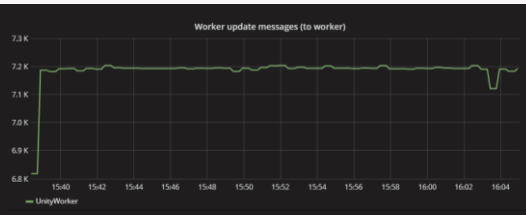
The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.8 625 cells simulated using a communication optimisation and a hexagonal layout

10.8.1 625 cells simulated across 4 workers

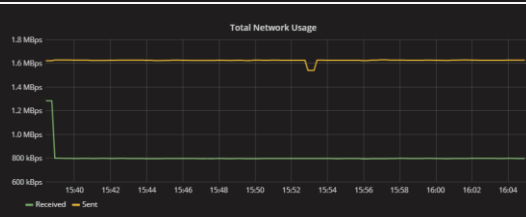


The rate of messages sent from an average worker to the SpatialOS platform per second.

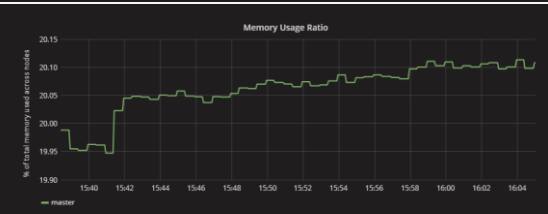


The rate of messages sent from SpatialOS to the average worker per second.

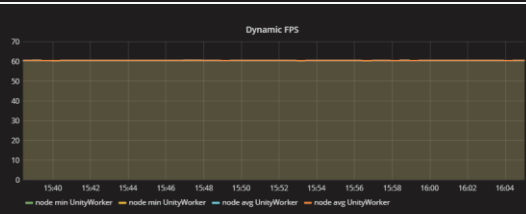
The message composition of messages sent to an average worker per second.



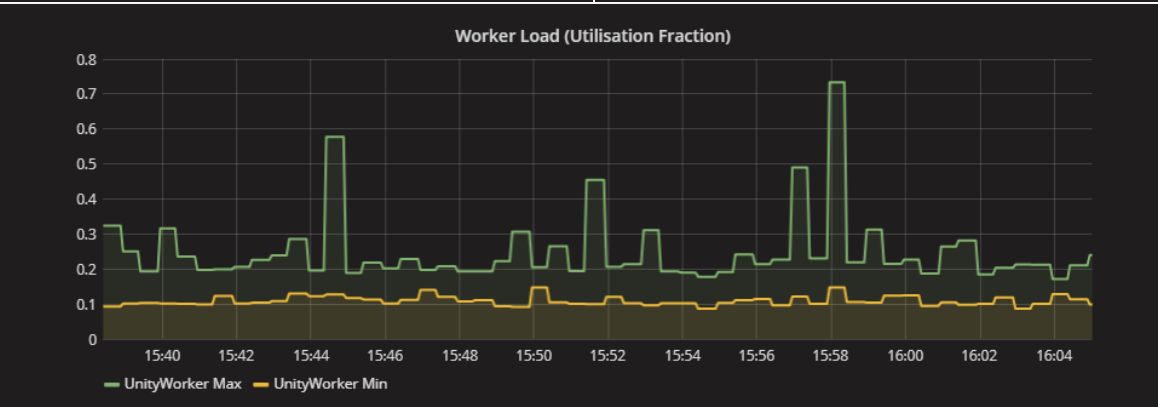
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



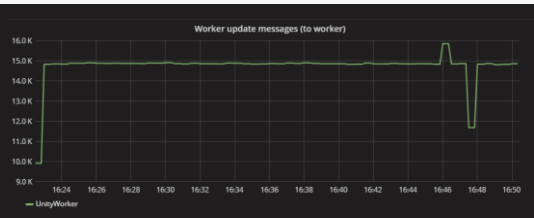
The average framerate of a worker.



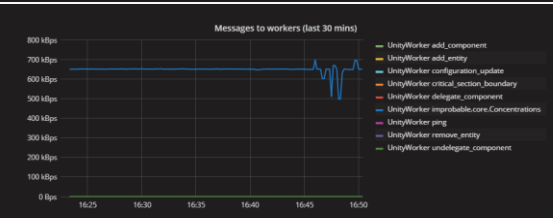
The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.8.2 625 cells simulated across 9 workers

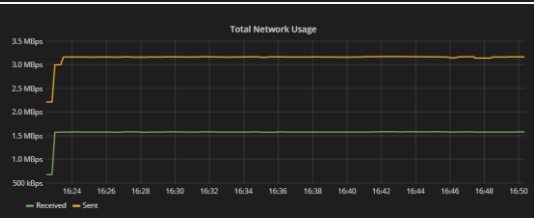
The rate of messages sent from an average worker to the SpatialOS platform per second.



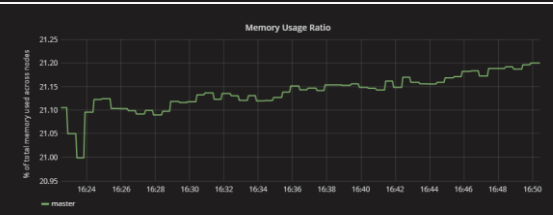
The rate of messages sent from SpatialOS to the average worker per second.



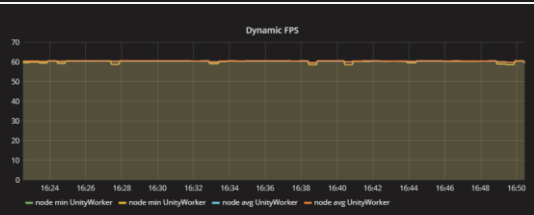
The message composition of messages sent to an average worker per second.



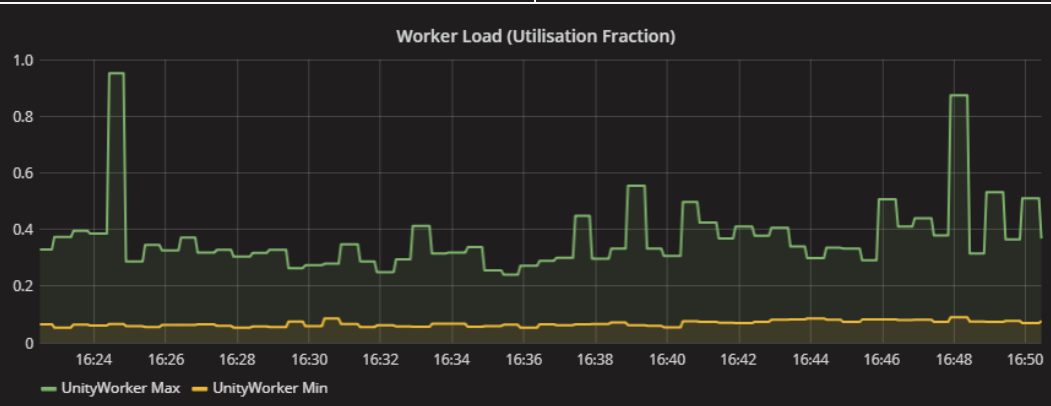
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.

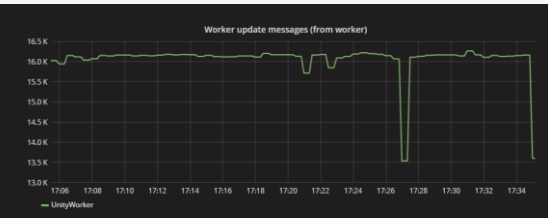


The average framerate of a worker.



The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

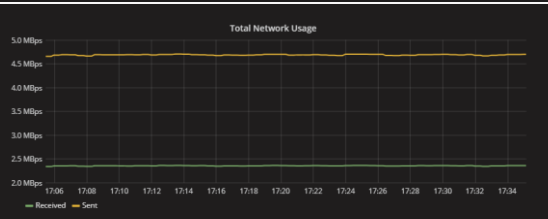
10.8.3 625 cells simulated across 16 workers



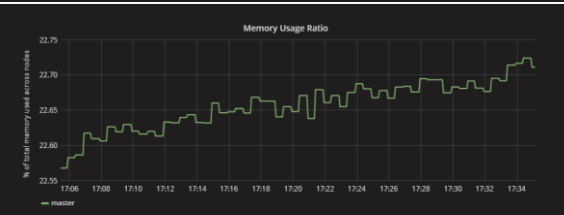
The rate of messages sent from an average worker to the SpatialOS platform per second.

The rate of messages sent from SpatialOS to the average worker per second.

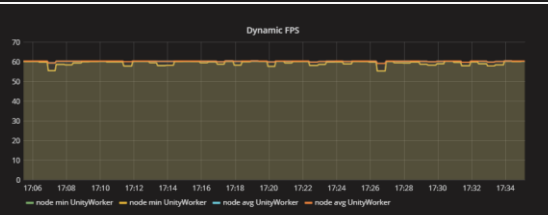
The message composition of messages sent to an average worker per second.



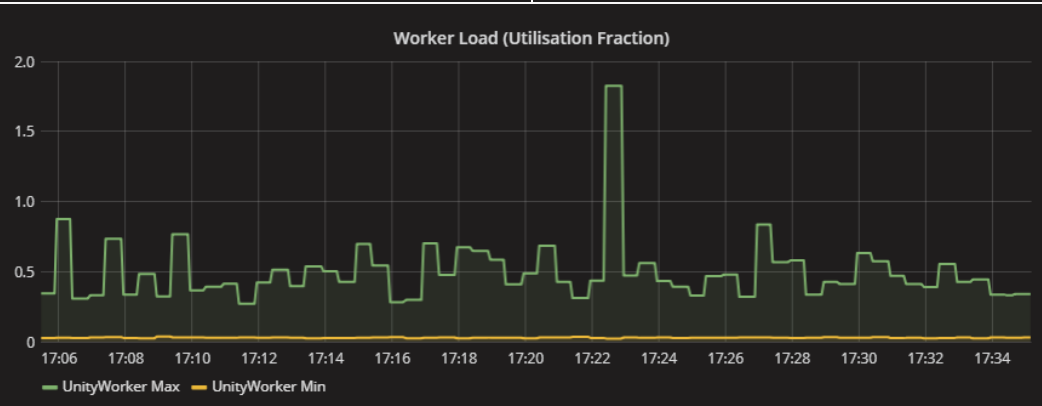
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



The average framerate of a worker.

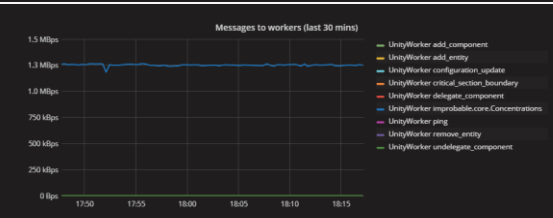


The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

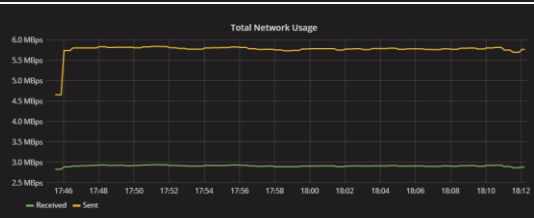
10.8.4 625 cells simulated across 25 workers

The rate of messages sent from an average worker to the SpatialOS platform per second.

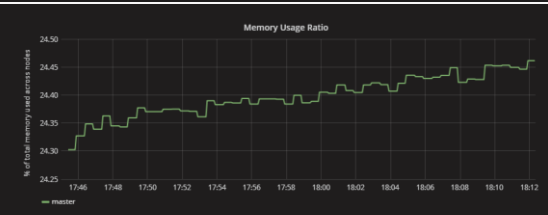
The rate of messages sent from SpatialOS to the average worker per second.



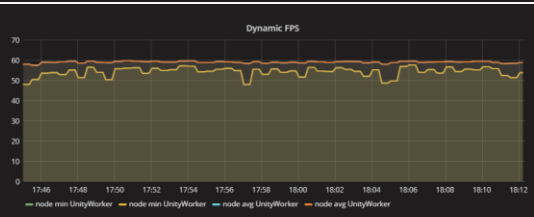
The message composition of messages sent to an average worker per second.



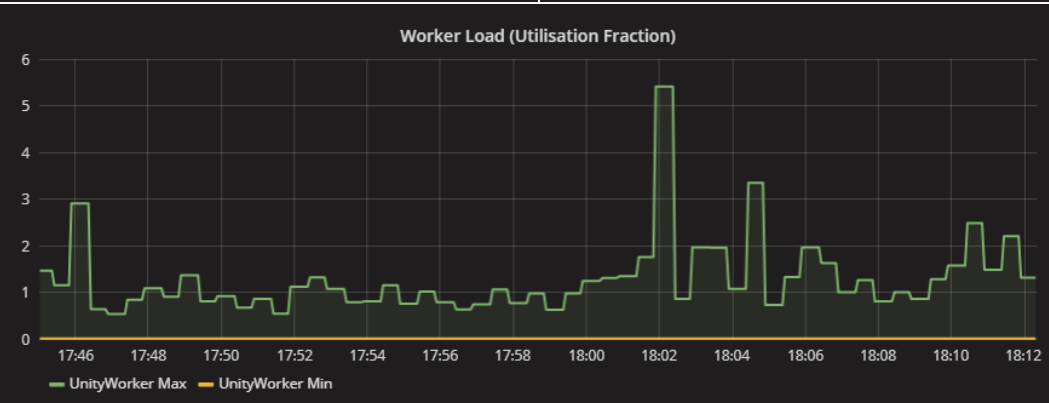
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



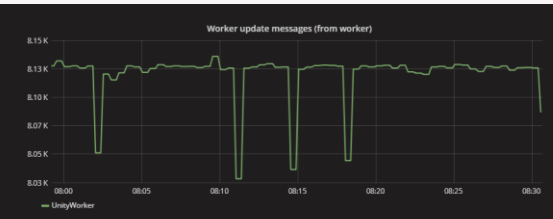
The average framerate of a worker.



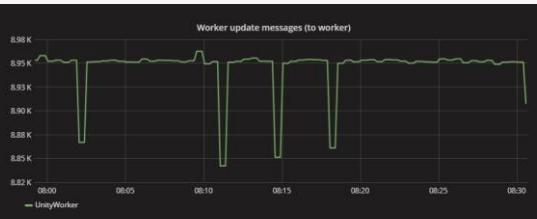
The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.9 900 cells simulated using a communication optimisation and a hexagonal layout

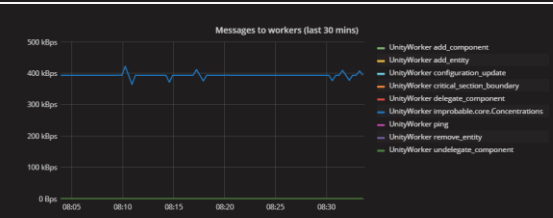
10.9.1 900 cells simulated across 4 workers



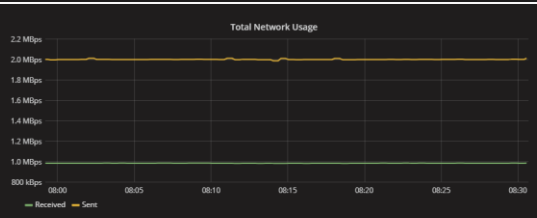
The rate of messages sent from an average worker to the SpatialOS platform per second.



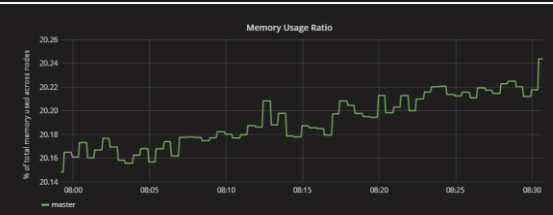
The rate of messages sent from SpatialOS to the average worker per second.



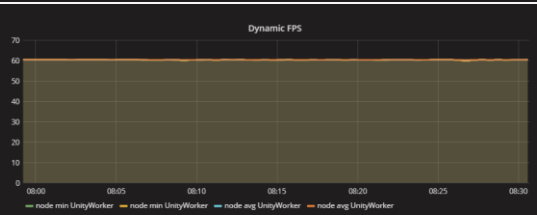
The message composition of messages sent to an average worker per second.



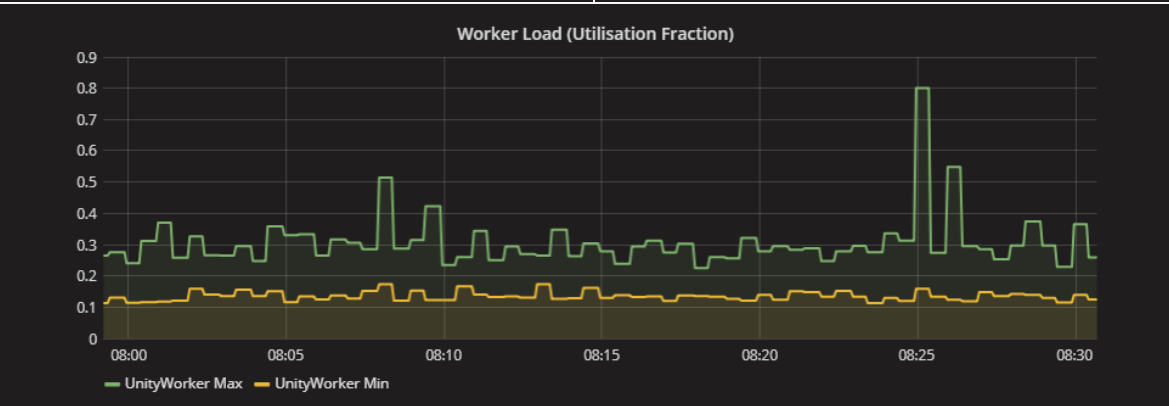
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



The average framerate of a worker.

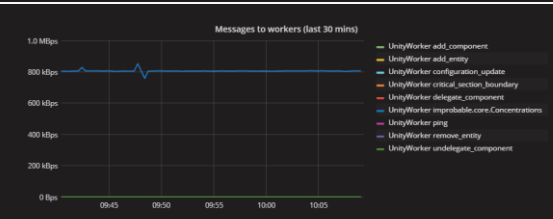


The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

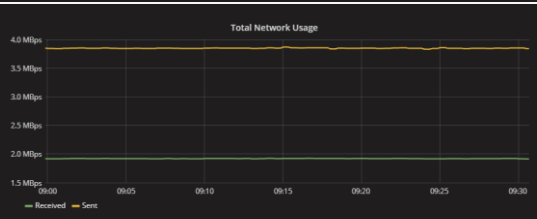
10.9.2 900 cells simulated across 9 workers

The rate of messages sent from an average worker to the SpatialOS platform per second.

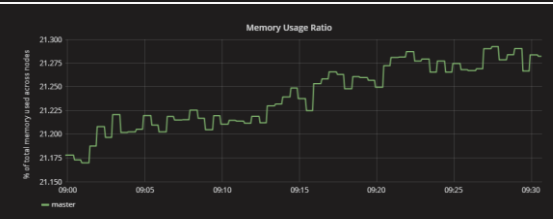
The rate of messages sent from SpatialOS to the average worker per second.



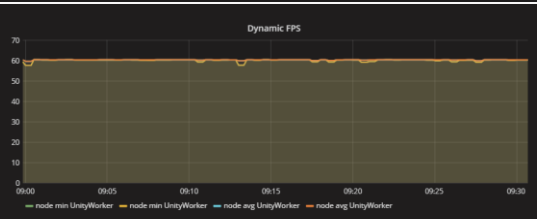
The message composition of messages sent to an average worker per second.



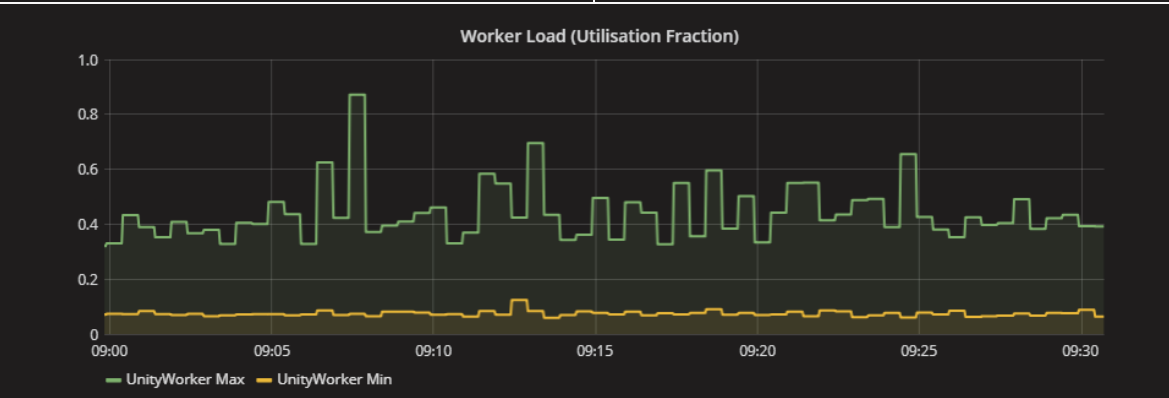
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.

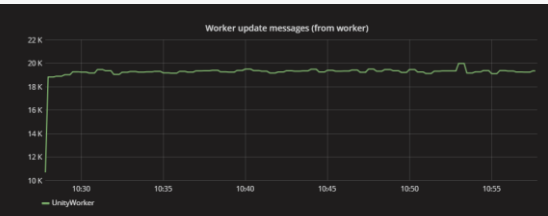


The average framerate of a worker.



The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

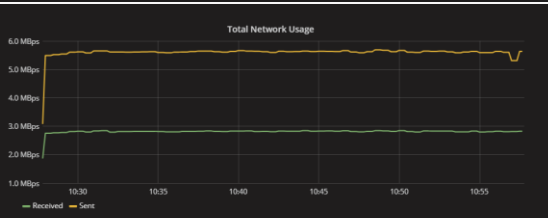
10.9.3 900 cells simulated across 16 workers



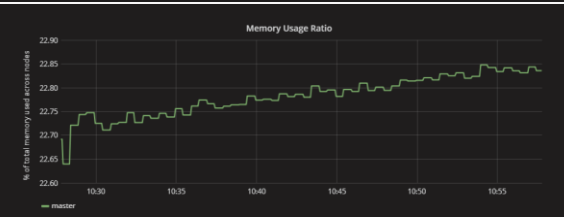
The rate of messages sent from an average worker to the SpatialOS platform per second.

The rate of messages sent from SpatialOS to the average worker per second.

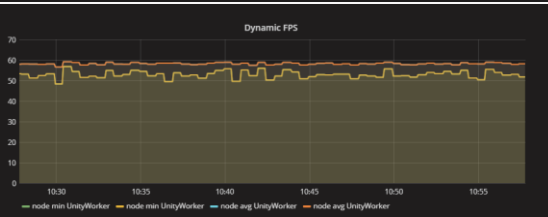
The message composition of messages sent to an average worker per second.



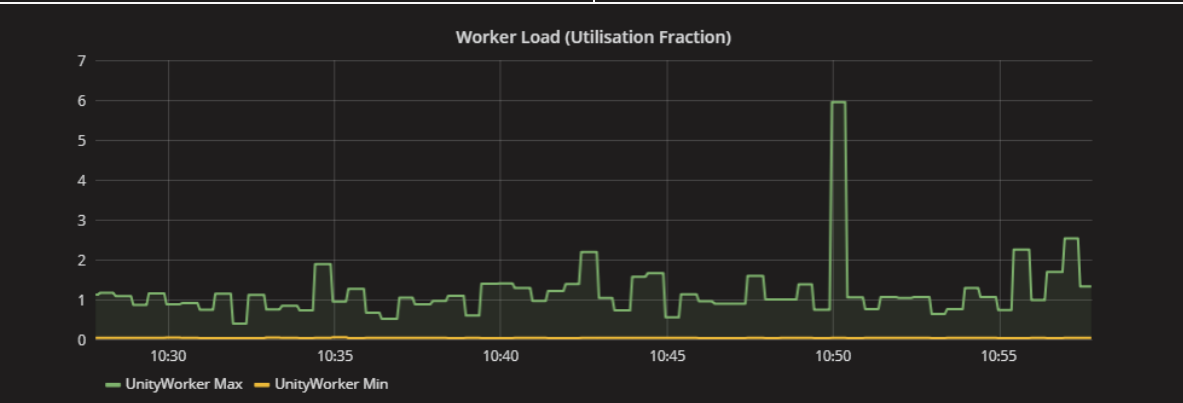
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



The average framerate of a worker.



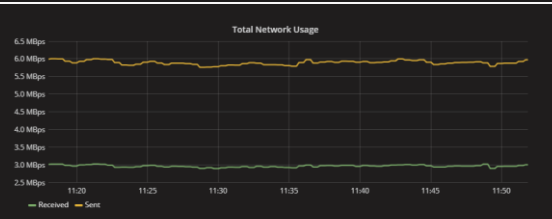
The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.9.4 900 cells simulated across 25 workers

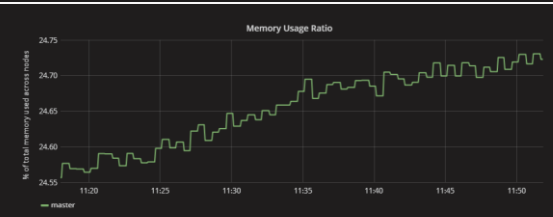
The rate of messages sent from an average worker to the SpatialOS platform per second.

The rate of messages sent from SpatialOS to the average worker per second.

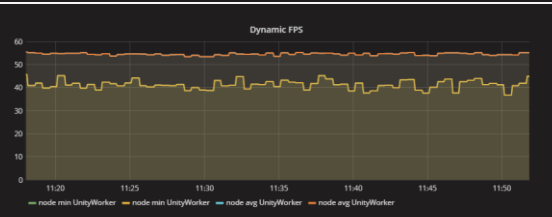
The message composition of messages sent to an average worker per second.



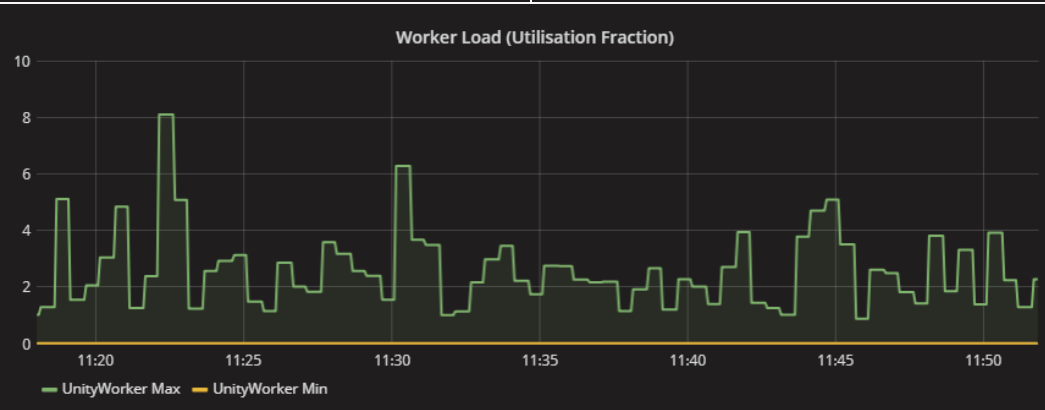
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



The average framerate of a worker.

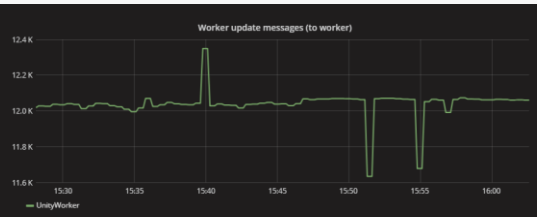


The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.10 1600 cells simulated using a communication optimisation and a hexagonal layout

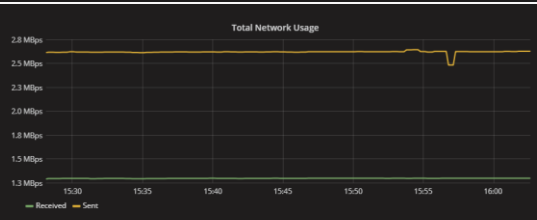
10.10.1 1600 cells simulated across 4 workers

The rate of messages sent from an average worker to the SpatialOS platform per second.

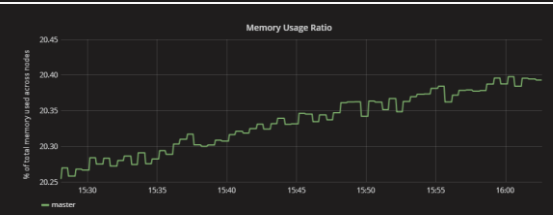


The rate of messages sent from SpatialOS to the average worker per second.

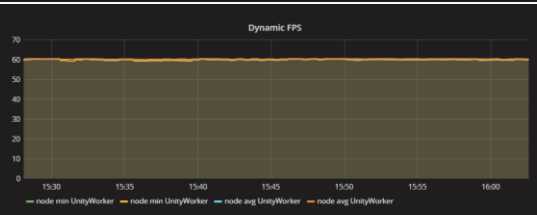
The message composition of messages sent to an average worker per second.



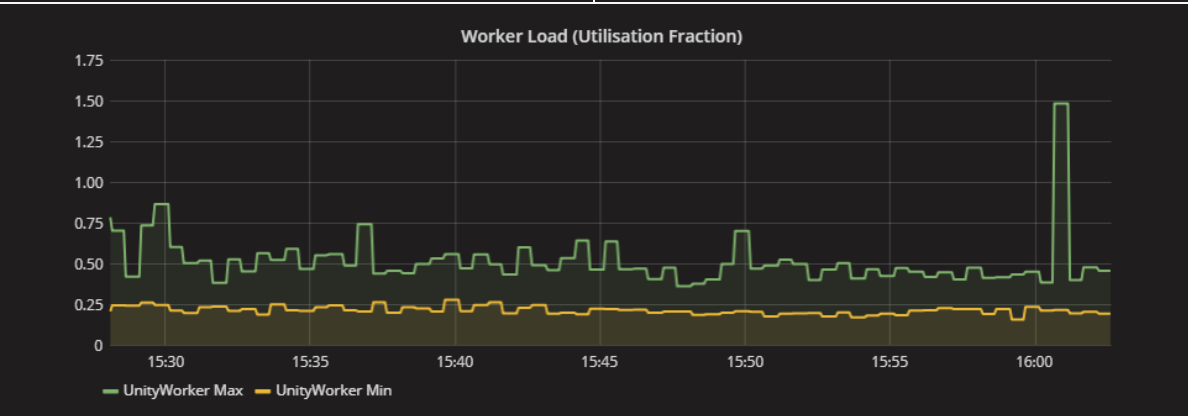
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



The average framerate of a worker.



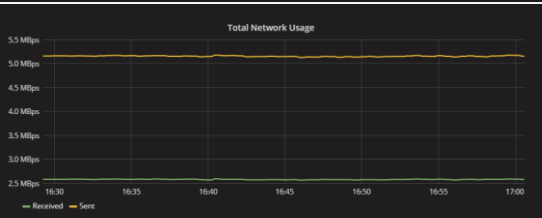
The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

10.10.2 1600 cells simulated across 9 workers

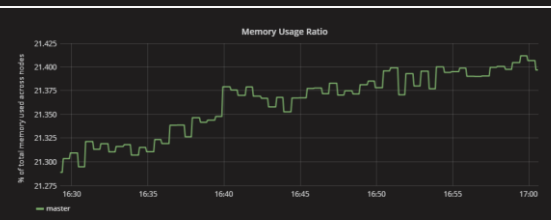
The rate of messages sent from an average worker to the SpatialOS platform per second.

The rate of messages sent from SpatialOS to the average worker per second.

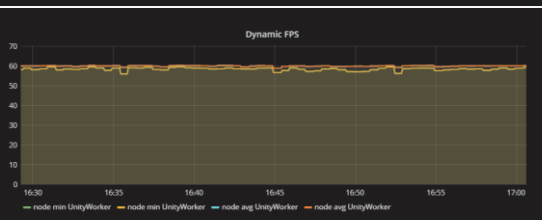
The message composition of messages sent to an average worker per second.



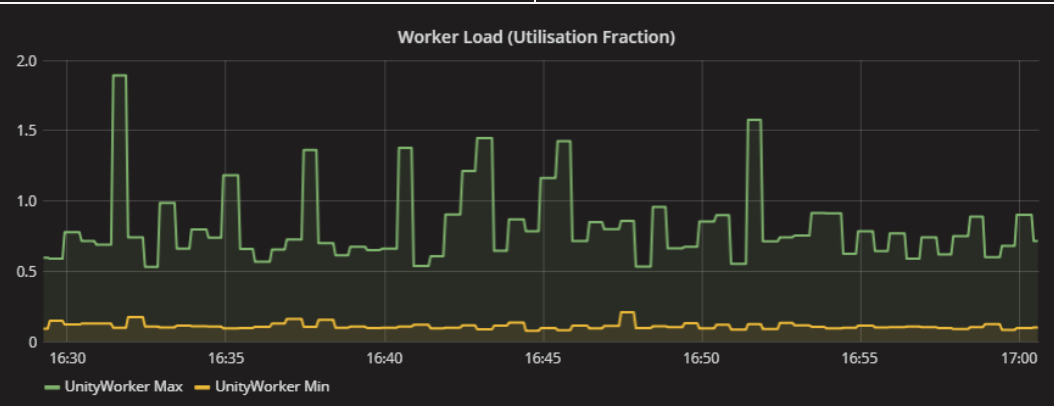
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



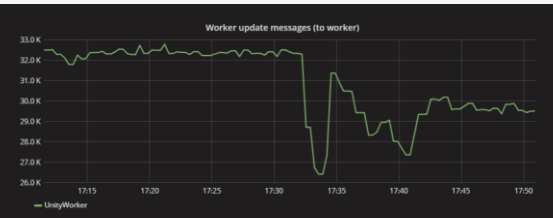
The average framerate of a worker.



The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

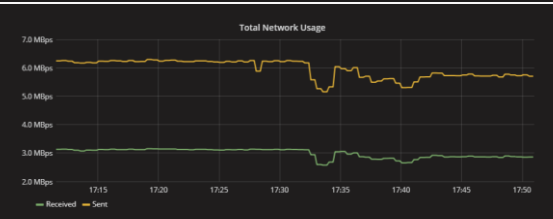
10.10.3 1600 cells simulated across 16 workers

The rate of messages sent from an average worker to the SpatialOS platform per second.

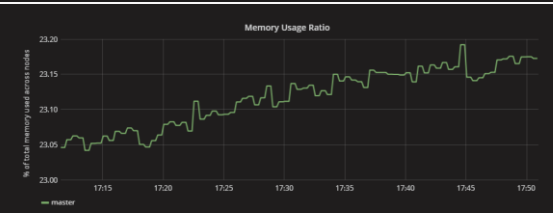


The rate of messages sent from SpatialOS to the average worker per second.

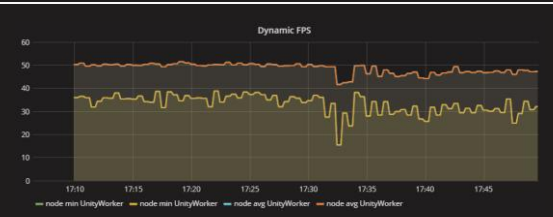
The message composition of messages sent to an average worker per second.



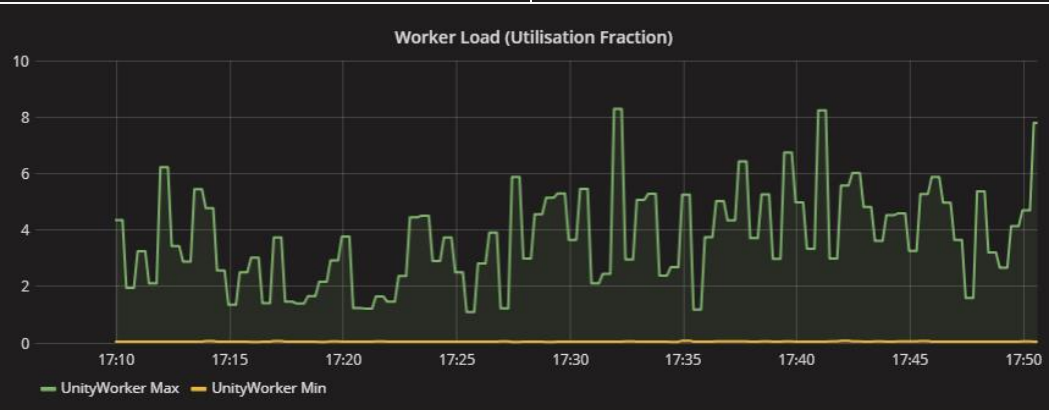
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



The average framerate of a worker.

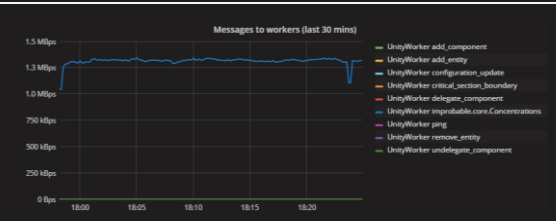


The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.

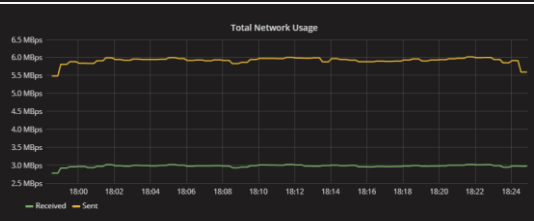
10.10.4 1600 cells simulated across 25 workers

The rate of messages sent from an average worker to the SpatialOS platform per second.

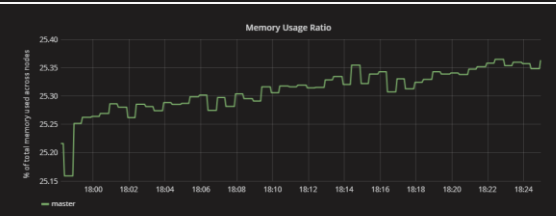
The rate of messages sent from SpatialOS to the average worker per second.



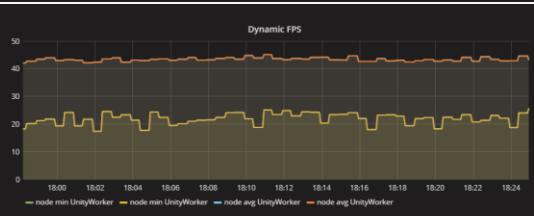
The message composition of messages sent to an average worker per second.



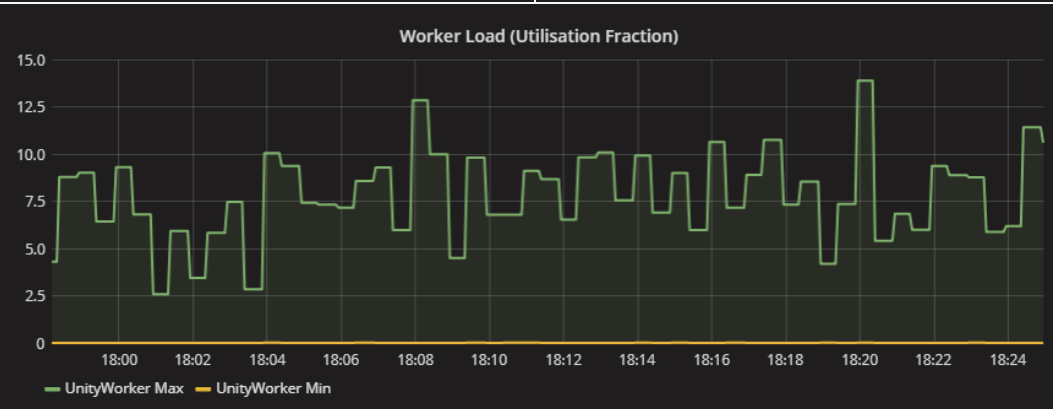
The total amount of data transferred over the simulation, grouped by data sent to workers and data received from workers.



The amount of memory used as a percentage of the total available memory available across the system.



The average framerate of a worker.



The load as a fraction of a workers total capacity, grouped according to the worker with the highest load and the worker with the lowest load at any time.