UCLouvain

epl

École polytechnique de Louvain

Leveraging edge proximity and diversity in interactive stateful applications

Author: **Guillaume Ortegat**
Supervisors: **Etienne Rivière, Jean Vanderdonckt**
Readers: **Donatien Grolaux, Guillaume Rosinosky**
Academic year 2020–2021
Master [120] in Computer Science and Engineering

Abstract

Context: Cloud architecture is optimized for stateful collaborative and interactive applications. Cloud facing high latency problems, edge computing is tested to propose the same level of service. The cloud presents only a single model that may not take into account the fact that users are close and share one or more devices.

Objective: The goal of this thesis is to implement a proof of concept to test the cloud to edge adaptation framework. The purpose of edge computing is to migrate the server part on the user devices. Leveraging low latency to implement user interface separation. We will evaluate the application with humans.

Method: We design the adaptation framework of a cloud interactive web application into an edge. We create multiple versions of the same application to split the user interface. We test the proof of concept in an experiment with two groups of five participants.

Result: Users show difficulty to differentiate if the application is using cloud or edge computing. They favor graphical user interface splitting on some, but not all, device types.

Conclusion: Edge computing is a valid alternative to cloud computing as it effectively addresses most of the the issues usually linked to cloud computing (latency and high network usage). The latency allows to detach graphical interface parts, which helps leveraging the characteristics of each device.

Acknowledgments

First and foremost, I would like to express my gratitude to my thesis supervisors, Prof. Etienne Rivière and Prof. Jean Vanderdonckt, for their precious advices. During the whole year, they responded with a low latency to my questions even during the weekend.

Thank you to the two readers of my thesis, Donatien Grolaux and Guillaume Rosinosky.

Many thanks to my flatmate for their support and rereading, Jean-Martin Vlaeminck, Arthur Sluÿters, and Maxime Mawait.

A special thank you to my friends and parents which participate in the experimentation and help me.

Without all their help and support, I could not have done this thesis.

Contents

Chapter 1

Introduction

Cloud technology is a part of our everyday life. If you go on the internet, you are bound to interact with cloud technology. Phones, laptops, watches, TVs, and many others devices have now become connected to the cloud, which would have been unthinkable 20 years ago. This became possible because cloud computing has been improved to be efficient and reliable.

Despite the efficiency and improvements of hardware and architecture of cloud computing, some limitations have appeared in the last few years. The growing diversity of user's devices and the ever-increasing size of the Internet of Things (IoT) are keys for understanding the desire to explore a new way to accomplish tasks done by the cloud [1]. Since 1999, the year when the concept of IoT was introduced, the success of IoT increased and with it the number of facilities proposed by the cloud. Multiple fields like healthcare, transportation, and many more use cloud applications [1]. In addition to that, the number of user devices never stops growing, those also using cloud applications. Humans using this application on these devices are sensitive to the user experience proposed. The interest in cloud applications and the possibilities proposed raise new claims and technical challenges to take up. One of the first difficulties to appear was the limitations of the hardware. The servers and big data centers were (and still are) able to process the amount of data, but the bandwidth to send the data to this structure struggles to transfer the data. Some solutions can be to compress or reduce the size of packages. Another solution is to reduce the distance of travel of packets through the network and thus reduce the congestion. This distance of travel is measured in the number of hops.

The increased use of cloud computing affects its performance. Thus, bottlenecks appear and can not be resolved by changing or upgrading software. Hardware has its limitations, and components of the cloud structure are congested or saturated. Without changing network usage, problems will not stop appearing and increasing. This is the cause of high latency which is a big issue of cloud computing. For this

reason, it will be one of the main concerns of this thesis. This must be our major concern to avoid performance losses and keep good quality of service for the user. Unexploited resources at the edge of the networks are available but applications are not designed to use them. Some solutions such as fog computing or mobile edge computing exist, but they have also issues and require new infrastructure.

The increase of diversity of devices creates new needs. The Graphical User Interface (GUI) must be adapted to the new possibilities of device characteristics. The evolution of devices and applications requires more than a single model for GUI which does not take into account the proximity of users and the fact that users may share devices with various characteristics.

1.1 Mission Statement

For all theses reasons, the issues of cloud must be resolved by an alternative. In these thesis, we will study the edge computing, one of the possible alternatives. We will focus on three objectives:
adaptation of a cloud computing interface based stateful application into an edge-computing based,
leveraging low latency of edge computing to split interface,
and *implementing a proof of concept and testing it in a real-life scenarios*.

Before diving into the subject, it is important to explain some concepts to facilitate understanding.

Edge computing is the core concept of this thesis. Many different definitions of edge computing exist. The common thread is that the server parts are moved closer to the users. We will give our definition of the edge computing concept. *Computing* is the operation to calculate or at least manage the server side of an application. This operation can be done by one or multiple hosts. In most cases, there is only one. In our definition of edge computing, the host is running on the user device, not on servers in big data centers. Our full definition of edge computing can be found in paragraph 2.1.1.

An interface is all facilities used to exchange information between parts of an IT system. The interface can be graphical, to exchange data between the device and user with button, text, ... but can also be technical with protocols and architecture to exchange between two equipments. Edge and cloud computing present two different technical interfaces. It is also good to notice that adaptation of the technical interface type can be more profitable to one or another graphical interface.

Real-life or concrete scenario is a proposed situation which can commonly happen in real life and not only in laboratory conditions. Concrete scenarios involve non-technical people using applications based on edge computing.

Stateful application uses variables and objects to store the actual state of the application. Many applications are stateful, but none of them is edge computing-based.

Now, we have a better comprehension of each concept of the mission statement.

This thesis focuses on improving latency and to move the server parts to the user devices at the edge of the network. There are some research questions we will answer:

- *What are the benefits of edge computing for adapted stateful applications?* This question is central to the thesis: it asks if the adaptation from cloud to edge computing is relevant from a performance point of view.

- *What are the perks of user device diversity for edge computing?* Edge computing-based applications must work with all kinds of devices. Service must be reachable and usable from all different devices. The efficiency of the adaptation depends directly on this parameter. To propose usage in concrete scenarios, this is a basic requirement.

- *Will the user experience of edge applications be at least as satisfactory as cloud-based ones?* This question is the focus on the user experience and the impact of the change from cloud computing to edge computing. Cloud being widely used, users expect a minimum level of user experience.

To answer this question, we made some assumptions. They are important because they help to fix the orientation of the goal. A first assumption is that the latency of edge computing applications is small enough to distribute GUI and fit to the device characteristics. We also suppose that all edge devices used can run classical web applications in a browser.

We will explain the conceptual adaptation of a cloud computing application insight to provide a proof of concept. This edge computing application will be used for experimentation with various people.

1.2 Overview of the Thesis

In **chapter 2**, we explore the existing information about edge computing in the scientific documentation. We will discuss the advantages and drawbacks of edge

computing compared to other computing paradigms. We will discuss the separation of parts of the graphical user interface.

In chapter **Chapter 3**, we explain the conceptual framework to transform a cloud computing based application into a edge computing based. We will focus on the points on which we have to pay special attention. We will describe the classical component of a generic cloud computing application. The different constraints faced. The adaptation of an cloud computing application into edge computing will be explained in sight of developing the proof of concept in chapter 4. The importance and the way to migrate the host will be demonstrated.
As interface are an important part of this thesis, we will see how to decompose the interface in "blocks" to create an adaptive application user interface. This one has a big impact on the user experience and is a key point to maximize the utilization of the capacities and advantages of edge computing context.
The self-sufficiency of the host and the interface will be highlighted. The importance of this point has to be understood to apply in the best way the previous modifications.

A concrete proof of concept is described in the **chapter 4**. The adaptation of the send-event-based architecture will be explained. A concrete application of the concept of chapter 3 is explained in this part.

In the **chapter 5**, we will test the hypothesis about latency and user experience with an experimental protocol. Despite the coronavirus, we were able to apply this protocol to two groups of five people. This experience aims to validate ours assumptions in the face of concrete scenarios.

We will conclude this thesis with **chapter 6** which will summarize the contribution and benefits explored. We will also discuss possible future works.

Chapter 2

Related Work

After presenting the objective of this thesis, we will explore the related works. In this chapter, we will expose the context where edge computing has been created and design.

First, we will define edge computing and explain the concepts needed to understand the comparison of different computation paradigms: the levels of the network and migration. We will expose the methodology used to discuss the papers and articles.

2.1 Preliminaries and Definitions

This section summarizes all concepts and definitions needed to understand the analysis of the literature.

2.1.1 Edge Computing Definition

Edge computing is used to describe multiple technologies reducing the time and distance travel by package between the server and user's devices by being aware of the localization of the users. In the literature, computer scientists are not according to the definition of edge computing. Some authors consider fog computing as edge computing [1], but others qualify fog computing and peer-to-peer not as edge computing [2]. In this thesis, we will use *Edge computing* to name applications that migrate the server host part on the user device. We will use and specify other technology's names which are in some papers also called edge computing. The edge computing paradigm is an architecture that is not user location blind and prefers to use the closest possible data center. In some articles, this is defined as a new way to shift the centralized function of cloud computing to edge devices [3].

The need to create this technology will be discussed in the next section results and discussion 2.2.

2.1.2 Migration of Data Used by a Service

The goal of edge computing is by definition to move the function from the cloud to the edge. Some solutions offer the possibility to migrate states between the cloud and an edge instance.

- to an edge server/device from the cloud
- between two devices/servers

The migration concerns functions but also data and state variables that need to be transferred. Two different ways to achieve are notable [4]:

- live migration from the data center is the operation set to move data from a machine to another. The data are transfer while the service is running.
- handover in cellular network is mainly used for mobile edge computing. Moving the service with few states and data, the cost of the transfer is light.

The main difficulty and requirements are to keep the live function working. Migration is essential and the needs of a project can change the choice of technology depending on the performance of the migration. Some criteria of the comparison in section 2.2 are direct links to the migration capacity and its performance. [4]

2.1.3 Depth of the network

To well understand the differences between some computing paradigm, we will separate the network into layers which we will call levels. We simplify the 5 levels of the network in literature [2]. The core level with big data centers is level 3. In level 2, there are modest data centers (distributed cloud) and the modular data center, cluster,... (level 2 and 3 in [2]). The level 1 correspond to the level 4 in [2] And level 0 to the level 5 in [2].

1. level 0 is the layer of user devices and IoT.
2. level 1 is the layer with gateways, access routers,... Closer than level 2 from level 0.
3. level 2 is the layer close to the user but in the network.
4. level 3 is the layer the most far away from the user, where big data centers are connected

An illustration of different levels of the network can be found in figure 2.1.

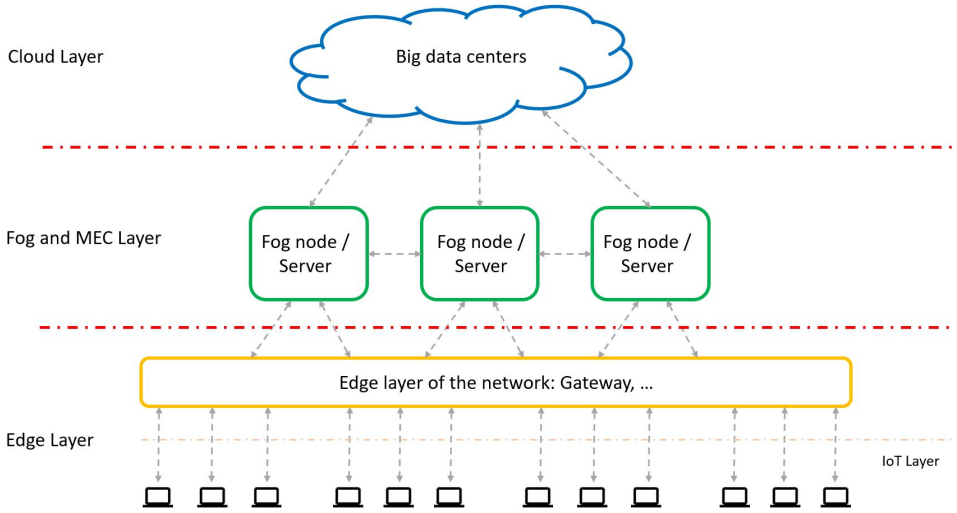


Figure 2.1: Proceeding layer of edge, fog, MEC, and cloud computing.

2.1.4 Method

During the information research phase, we found lots of cloud computing alternatives. We will first present them.

The best way to understand all the differences between each technology presented is to compare them to have a global overview of the benefits and drawbacks of each.

First and foremost, we will explain all elements compared to understand the strengths and weaknesses of each.

The base of this comparison is cloud computing for two main reasons. First, this is the most widely used in the world but also because most of the solutions presented are solving one or multiple problems, in some less wide context, of this technology. Used for few decades, despite the evolution, the cloud can not fulfill the specific characteristics of some application requirements. The comparison will be introduced by a table that resumes the point of comparison. After we will elaborate on some criteria.

2.2 Results and Discussion

2.2.1 Interface Detaching

Two of the key benefits of edge computing or other technology bringing nearer of the user the server parts is to decrease latency of delivery and increasing data rate [5]. Edge computing is also designed for human devices which have great diversity. Many kinds of devices bring lots of different characteristics, Different user interfaces and technical improvements lead to a new possibility as detaching interface to propose more flexibility. Detaching part of GUI using peer-to-peer fashion has already be done [6] [7].

Moving a Graphical User Interface (GUI) requires migrating some parts without interrupting or modifying the service, migration 2.1.2 is important for this ability. Moving GUI involves also splitting, detaching, grouping interface of parts of it. To match the needs of the users, the Detachable User Interface is a new paradigm tested in multiple scenarios (one screen to many screens, single or multi-user, single or multi-threaded) [6]. GUI is not linked to a specific device but can be adapted to user needs. Edge computing offers a better user experience of detached GUI mainly links to the low latency of the response perceived [8] by the user.

Edge computing allows deploying peer-to-peer distributed GUI across 4 dimensions: multi-display, multi-platform, multi-operating system, and multi-users [7]. Peer-to-peer is one of the most basic designs of edge computing serverless. Cloud applications present too large response delay to propose a good user experience with a distributed user interface. [7]

2.2.2 Cloud Paradim Alternatives

Conventionally, Edge computing can be considered as a substitute for cloud computing. This is not the only other technology existing, and this are the computing paradigms we will compare:

- Cloud computing [3] [5] [9]
- Edge computing [9]
- Fog computing [3] [5] [10] [1]
- MEC (Mobile Edge Computing) [3] [5]

Cloud computing is the most widely used computing paradigm on the world wide web 1.1. The request starts from the client and travels across the internet network

to reach a location blind big data center. The illustration of layer of the network used can be seen in Figure 2.1
The Security and performance of servers have been improved [11] with new architectures. Monolithic architecture with function calls everywhere was dropped for incremental structure in layers. The last improvement is the utilization of micro-services to encapsulate different features in a little service that is independent and can communicate with others. This optimizes scalability and maintenance.

Despite this improvement, computer scientists are facing hardware limitations. Packages have to travel from the user to the server. Migrate cloud services closer to a user is the best way to reduce the communication latency of the response. All following technologies propose different ways to reach this goal.

"*Fog computing* is a distributed computing paradigm that provides cloud-like services to the network edge." [1] We can consider it as a combination of cloud and edge computing paradigm [3]. It doesn't use location blind servers but installs and uses smaller servers in level 2 of the network to be closer to the final user. The hosting role is never migrated to user devices. It is has been developed also to be a good solution to manage the big amount of data generated by IoT [10]. Packages will prefer, if possible, to not go upper than level 2 (Figure 2.1).
The name fog computing is not due to a personal choice of computer scientists but is a picturesque manner to describe the main idea. The used server is closer to the user (and to the ground), the application will prefer to use fog instead of the cloud for the proximity advantages. The levels where the packet travels in edge computing can be seen in figure 2.1.

MEC or Multi-access (or mobile depending on the paper) Edge Computing [3] is developed for specific use with a mobile network (network for smartphone, and portable devices). This paradigm looks like fog computing except that it is optimized to follow moving devices as a smartphone or specific IoT devices. The design goal of MEC is to provide the best connection even you move. The optimization of the migration is the key point of MEC and the main difference with fog computing. MEC is using an edge server (smaller datacenter), but unlike fog computing, they expect high mobility of the user. Fog computing can offer this mobility due to the close range of efficiency of edge servers without modifications. These changes create MEC that is used for mobile data.
The main advantages of MEC are:

- To reduce the network operational cost
- To improve Qos (Quality of service)

- To improve scalability of IoT for time-sensitive applications

The differences between MEC and fog computing are not sufficiently large to use them both in the comparison. So we will consider only fog computing in the comparison. Other paradigms also exist and are close to MEC or fog computing (e.g.: Follow Me Cloud paradigm) so will not be considered in the following section.

Understand main differences between edge, fog, MEC and cloud computing A good way to reveal the difference between these 4 paradigms is to look at the layer where they proceed [12]. The illustration in the figure 2.1.

Fog and MEC operate on the same layer, but the fog is static and migration is not at the core of the paradigm. MEC is done to give the best access and speed even if the device in the IoT is moving (smartphone for example). MEC will use also other networks in some configurations such as 4G network.

2.2.3 Comparison of Computing Paradigms

We can compare these computing paradigms on many criteria. We will compare cloud computing, fog computing, and edge computing. MEC having performances comparable to fog computing.

We will begin with the criteria linked to the network architecture. First, the latency of the paradigm. Edge present the best latency using peer-to-peer [6] [7] technology to exchange messages. The packet uses a straightforward path between devices. The hosting part being migrated on the user device, the travel latency is reduced to the minimum. The travel latency perceived is classified in the fastest category [8] for humans: the action seems to be immediate. However, global latency is travel latency added to computational latency. To see if global latency is still in the best category, we have to analyze the hardware power of devices. The diversity of edge user's devices don't ensure to have enough computational power to offset the large computational power of data centers. Hardware capabilities are still improving following Moore's law [13]. Except for specific high requirement applications, edge computing can split the computational task between multiple devices or evaluate the best equipment to run the task using a mathematical approach [14]. Edge computing presents the best global latency. Computational latency for light and medium applications doesn't have a significant impact on global latency. For big hardware requirements, there are too many variables to decide if the global latency is impacted. Edge computing has in most cases an excellent global latency.
Fog and cloud computing paradigms use dedicated servers, so the computation hardware latency limitation is negligible and the travel latency gives the following result: Fog does better than cloud. However, they are both beaten by edge.

Connection reliability is one of the key points of edge computing. This paradigm can run services without servers (except the connection part) even if network fails. In case of host failure, another user connected device can replace it with migration in the distributed computation network. Fog computing presents also good connection reliability [15], packet trip in the network structure is short and using fewer levels and infrastructures. Cloud can be unreachable in case of global or local network failure, or if the server fails. Thanks to cloud connection failure prevention, disconnection is highly unlikely [15] [16]. The downtime is the interruption time in case of updating or failure leading to stop service. Due to the distributed network of fog and edge computing, downtime of service is short or impossible. The cloud data centers are highly-reliable and highly-redundant so if a data center fails, big applications will be host on other data centers. Downtime is negligible for all paradigms.

Security and data management are different topics but the reasons for their performance are similar. The shortest path used for limited data transfer in edge computing allows great security and reducing data transfer simplifies data management [17]. No transfer of data to the cloud is needed [18]. Even fog computing using servers, the distributed cloud used to dispatch information across a huge number of node ensure data protection [15] contrary to cloud [19].

We compare also the three computing algorithms on computation power traits. Processing big data and using large-scale applications are both depending on the scalability proposed by the paradigm. Edge computing has a big computation power available, but it is considerably distributed and the total amount of computation power can have big variations. Good edge scalability is still a challenge even if improvements are in progress [20]. Big data in edge are rare because the data are processed locally and try to avoid the accumulation of data not processed. Edge computing design is not optimized for big data. Large-scale application performances depend on the separation of the task. Clouds present excellent capacities for computation power thanks to big data centers. However, the scalability of the cloud is limited by the bandwidth hardware limitation unlike fog and edge computing [1]. Fog computing has the same capacity as edge except for the usage of distributed computational power across multiple servers at level 2. The variance and scalability are better due to this optimization of the paradigm.

2.2.4 Edge Device Computation

The diversity of edge devices is considerable and multiple parameters have to be considered to distribute the computation task. Edge/cloud hybrid paradigm

propose solution to take in count cost for computation tasks link to many parameters [21] [22].

Adapting cloud application is not an easy task. A monolithic architecture will complicate the job and ask for more work to refactor the structure of the application. Micro-service is a modern structure that is already split into multiple parts giving high modularity. Each microservice has to be adapt and evaluated for its mobility [22].
To ensure the consistency of stateful applications, a framework as Legion exists [22]. It does not require significant modifications and is based on the Google Drive real API usable for peer-to-peer connections. It shares the state of the application to avoid losses.

For the moment, an example of cloud application adapted in edge computing exists but only for static applications. Facebook has proved that it is great for commercial use by adapting some of their applications [23].

`Edge devices have limited` computation power, offloading libraries exist to offload big tasks [21]. CPU discharges the more battery, for mobile devices, using a service must not drain all the battery. They incorporate a decision subsystem using a cost model which will evaluate the task and helps to decide the best user device to compute it. For example, A big computation task is preferable to be executed on a desktop instead of a smartphone, even if the travel latency is bigger.

2.3 Conclusion

The emergence of new needs which can not be fulfilled by the cloud technology due to hardware limitations, especially the bandwidth, leads to new computation paradigms. Fog, edge, and MEC are part of them. The comparison shows that the usage of each of them responds to special requirements. MEC was created to optimize mobility and offer the best access point with low latency. Fog computing uses a closer server and is location award. Edge computing is an alternative to fog computing, trying to stay at the closest levels to the user. Distributed cloud (edge, fog, and MEC) present good advantages as security, data management, and overall low latency. This low latency offers a new possibility for detached interfaces.

The advantage of edge computing is to propose high security by reducing latency and data transfer. The proximity to the data source also facilitates data management despite lower computational power.

Chapter 3

Conceptual Framework

In this chapter, we will discuss the adaption of a cloud computing application to edge computing. The goal is to leverage the computation power of user's devices. The application is stateful, interactive, and collaborative. We will focus only on the ones used in browsers. Leveraging the low latency of edge computing, we will split the user interface. This modification aims to fit the device characteristics. This is done in light of developing the proof of concept in chapter 4. First, we will explain the different characteristics of a cloud application and the wanted design of the edge ones. After that, we will discuss the important points of the adaptation.

3.1 Overview of Cloud and Edge Application

In this section, we will review the design of cloud and edge applications.

3.1.1 Cloud Application

Cloud applications present two big parts: the server and the client parts.
The server is used for multiple tasks. It will send the scripts and files of the application to the client. It will exchange packets with the clients to ensure interactivity. It will also keep the state of the service for stateful applications. We are using this kind of application in this thesis. The server is located in a big data center.
The client is the device of the user. The client will connect to the server. First, he will execute the connection mechanism to start the service. A mechanism is a set of operations to accomplish a task. Continuation depends on the application type. The client is composed of two parts: the frontend and the backend.

The frontend is all the functions linked to the interaction with the user. The data display is also included in this part. For example, a button on a web page

and the function executed on his activation belong to the frontend.

The backend is the management of the local behavior and the interaction with the server. Computing tasks and packet exchange are part of the backend.

Figure 3.1 shows the different components of the cloud computing architecture. The server exchanges information with the client device thanks to the backend of the client code. The client interaction with user are ensure by the frontend part of the code. The proof of concept developed uses event-based mechanisms

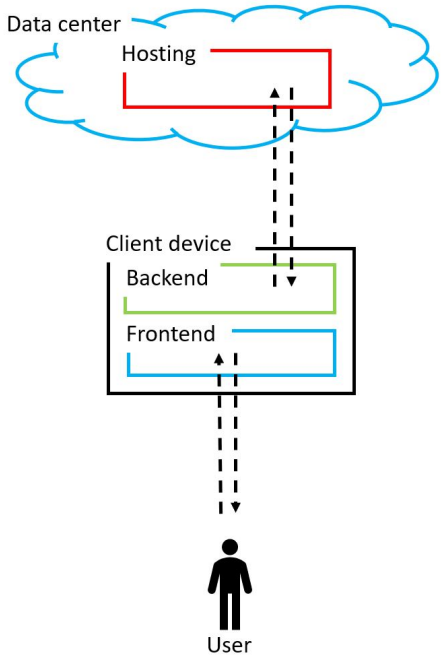


Figure 3.1: Illustration of the different parts of the cloud computing architecture.

to communicate with the server. This is a classical way to exchange information between client and server. In Figure 3.2, we can observe a client sending a packet with an event-based protocol. Each event has its own unique name. In the packet, in addition to the name of the event, the client or the server can add an object or a variable to exchange data. For each received event, a function is executed by the receiver. This is the base principle for the communication of cloud applications.

3.1.2 Edge Application

In our definition of an edge computing application, the server parts are migrated to the client-side on the user device. The edge goal is to be user localization

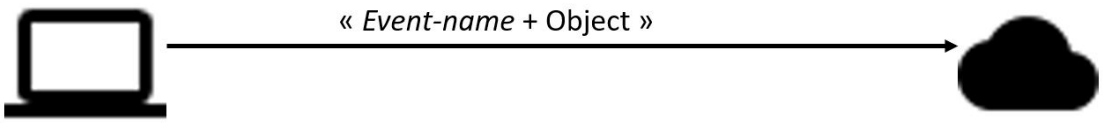


Figure 3.2: Event-based protocol example. Client sending packet to the server.

aware, to reduce the latency, and to propose multiple user models to fit device characteristics. Edge computing is an alternative to cloud computing but is not serverless even the hosting part is moved on the user device. A server is still used to send the application files to the client. It will also help clients to be connected in a peer-to-peer way. It can have a temporary hosting role if the context is not favorable. For example, the connected user device is not powerful enough. The hosting functions can be split to take advantage of the power and diversity of edge devices.

Migration of the hosting part has a central place. For multiple reasons, the hosting parts can be migrated during the execution. The cloud server functions can be moved to user devices in different ways.

Other client parts used in cloud computing are also present in edge computing. The backend and frontend can be grouped on the same user device or can be split on different devices. This separation is useful for user interface separation.

3.2 Adaptation from Cloud Application to Edge Application

The adaption of a cloud application to an edge application requires adding a host migration mechanism. The separation of the hosting parts can be useful to fit different computation application requirements. We will also add mechanisms and modify others to implement the edge version of an application.

Separating the graphical user interface will leverage the diversity of edge device characteristics.
One of the adaptation goals is to limit the increase of complexity due to the modification.

3.2.1 Main Constraints

The first step is to design the generic framework and respect constraints to fit concrete scenarios. Some requirements are essential, they can be resumed to not change the user habits. Based on this constraint, we can elaborate on four constraints. The first is to keep the same affordability of the application. The second is that the application must run in a browser on the user device. The third is to keep the information exchange architecture of the cloud. The last is to keep the same behavior as the cloud application from the user's point of view.

Cloud applications are reachable with HTTP protocols. The URL of the application is translated by DNS resolvers to get the IP address of the server and connect to it. After the connection is established, the server and the client will exchange packages. From the user's point of view, he just enters the URL of the application and the connection is established with the server. Changing the computing paradigm must not change the way to use applications. Our solution must propose the same affordability behavior from the user point of view.

Classical browser software such as Google Chrome, Firefox, Safari, Vivaldi creates the same constraints. Browsers are not able to use directly external libraries as cloud servers would do. `Require` or `import` commands will not work in a browser and need adaptation.
Cloud server uses multiple libraries to implement an HTTP server to send and receive packets. Browsers can not make run server and anyway it won't be efficient. An alternative must be used or implemented.

Devices using the cloud commonly use event-based communication with the client. proof of concept will use event-based communication as explained in paragraph 3.1.1. The client or the server sends a message with an event name and optionally variables or objects in the message. The receiver of this message will have different behavior depending on the event and the content of the object. Computer scientists are trained for event-based interaction styles such as REST and keeping this mechanism is a good thing.

In addition to the classic browser software and the way to connect to the service, other behaviors have to be preserved.
Synchronized data between users by the server is common in applications, the efficiency of this ability must be conserved.
Fluidity and continuity of the service are essential. Host migration can not disturb and must keep the same level of performance.

3.2.2 Migration of the Host

The host part can be migrated in different ways:

- Starting the hosting part(s) directly on the edge device, on the first connection, or to initialize a new room
- Live migration of the hosting part(s) from the server to the edge user
- Live migration of the hosting part(s) from an edge device to another

A starting migration of the hosting part(s) directly to the edge device is the easiest one. The starting state is generally empty or set by default. Initiate the server-side on the client device is easy: just warn the user device that he host and no state has been already set. This kind of migration is not different than the migration of static applications.

A live migration during the application is running is harder and is important for the stateful applications. The state variables of the application have to be transferred without interrupting the service and without losing any information.

Information losses can happen if a new information modifying state is sent to the old host and this one has already transferred the states to the new one. Synchronization between the two hosts is a key point for the success of the migration.

In the proof of concept, we will use both migration types.

3.2.3 Host Function Division

The migration of host functions uses different mechanisms. Different applications have different computational power requirements. The diversity of edge user devices can not ensure to have one powerful device to host the whole service and simply replace the server. Some applications need more resources. The hardware on each user device is limited and the code has to be adapted to it. For example, preferring loading non-mobile devices with heavy parts to avoid discharging mobile device battery. To deal with the diversity of device, four different computational tasks separations can be done:

- *Split-once* : Multiple devices host each a part of the application
- *Split-multi* : Multiple devices host each part of the application and each part can be hosted at the same time on multiple devices.
- *Whole-for-one* : One device host the whole application

- *Whole-for-multi* : Multiple devices host the whole application

An application will not use all four ways to achieve this. Depending on the service requirements, some separation can be preferred to optimize performances. We will explain the four separation methods and we will see which application requirement fits best with each method.

Splitting of the Hosting Task

Split-one is a simple way to distribute the computational task. To use this method, the task must be split into multiple smaller tasks. If it's not possible, this method and the split-multi method are not applicable. The split-once application will delegate parts of the hosting role to multiple devices. In this thesis, we use stateful applications and they require sharing states about services. Consistency of the state between parts is really important. To not complexify the implementation, gathering all states on one device is easier.
This implementation can be useful for high-performance applications requiring big power computation such as a web application of 3D simulation of custom shapes which can be edit by multiple users.

Split-multi requires a splittable application server task. Split-multi can be used for applications using multiple rooms of users. The same sub-tasks can be hosted multiple times on multiple devices. For big stateful applications, it is the most complicated because the state must be shared between devices hosting state variables. Some papers propose solutions, but this thesis is not focused on the scalability of big applications. But on more "classical" applications. We will only consider application using rooms of users for split-multi, which means that is comparable to have multiple instances of the application which are split-once and are not related between them.
The application for 3D simulations creating one split instance of the application for each simulation is a good example.

The computational task of the proof of concept is quite lite, we will not choose to split the hosting task.

No splitting of the Hosting Task

Whole-for-many has two use-cases: the hosting part can be hosted by multiple devices to be fault-tolerant and ensure the service. In this case, the state must be shared between different hosts. The second case is if we have multiple instances of the application, for example, games online, the states of each playroom are independent and hosts are independent of each other. The whole application is

hosted multiple times but the state of each instance does not have to be shared with other instances. A good example is the only document modification, for each document a new instance of the application is started. To keep a simple implementation for the proof of concept, we will not use this.

Whole-for-one is the simple way to host the service for client devices. It needs less adaption work and simplifies the design of the adapted application. It consists to not share the hosting task and replace the server with only one edge device. For small applications, this is a good solution.
Application with low computing requirements as Pictionary games is a good example. We will use this mechanism for the proof of concept.

To summarize, splitting the host task can be useful to deal with low computational devices or applications with high computational requirements, but it complexifies the implementation. Hosting the whole application on one device is easier but requires to have light computational requirement application or user edge device with good performances. In the next chapter, we will present the implementation of a proof of concept using whole-for-one hosting separation.

A graphical representation of the four method is represented at figure 3.3. In this representation, the client devices are red rectangles, hosting tasks are blue rectangles. When the hosting task is split, letters in the blue rectangles represent the different parts. The same letters show the same parts.

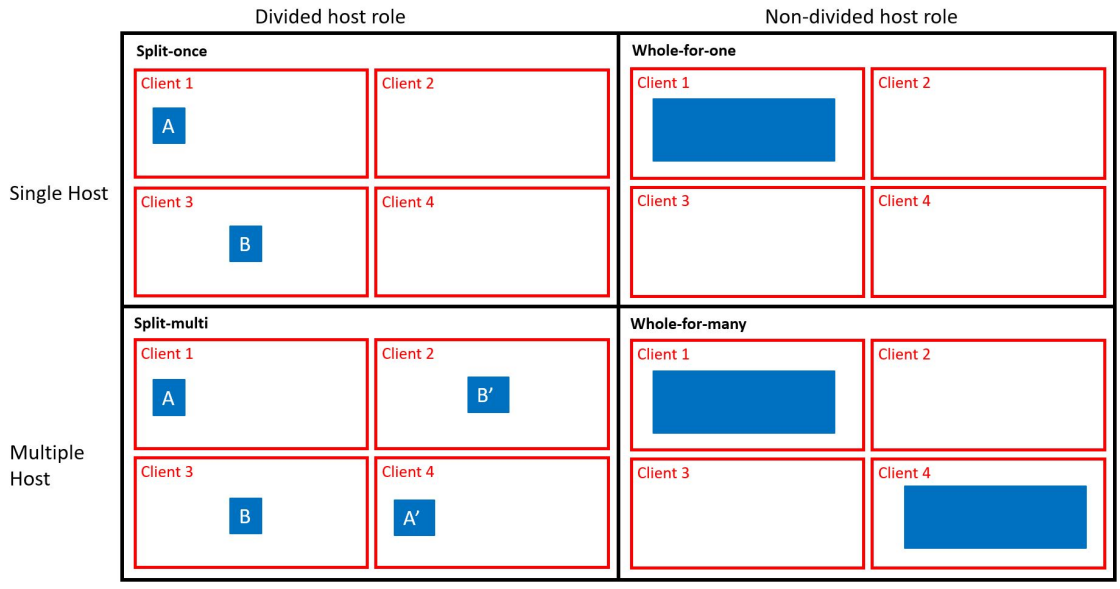


Figure 3.3: Four separation of the hosting part method summary.

Service State Consistency

Service state consistency is really important. Stateful applications are always using state variables and bad state management will lead to an interruption or a failure. Library designed to resolve this problem exists using online storage (see chapter 2). In our solution, the server used for discovery may be used to save the state in case of massive device failure or disconnection. But the most efficient solution and quick to recover from a failure is to share the state with all devices even they are not hosting. In case of failure, another device can host the service without interruption.

For basic application which used same state variables during a short time and are more state losses tolerant, the state has not to be shared with another device than host. But this solution is not protected again failure and the state recovery is harder. The only advantage of this is the easy implementation. The proof of concept of chapter 4 being state loss tolerant, we do not add a state loss protection mechanism.

3.2.4 Adaption to Edge Computing Structure

Moving the hosting part to the user device requires modification of the code. The constraints in paragraph 3.2.1 are motivations of some choices for the design of the solution.

Server and Connection of Users

The user of the application does not have the script or any native way to connect to the edge application. We will use browser or cloud protocols. To deal with the constraint of application affordability and the client connection, we use a small server in a data center to help with this task.
This server has two main functionalities to ensure. Firstly, send scripts and files useful to the client. Users will connect to the server address as they do for a cloud application. The second role is to help and facilitate the peer-to-peer connection between user devices. The discovery of other users is a key point for edge computing. The peer-to-peer connection mechanism is illustrated in figure 3.4. It illustrate a device wanting to connect will be connected to a server which will help to connect to others devices. Depending on the application, a user device can be its host or can wait for other users. The first host migration is done from the server to the user. If the device can not support the rise of computing charge of the hosting (which can be easily measured), the server can assume the role of the hosting. The solution in chapter 4 being a Pictionary, the game will begin if at least two users are connected. The server will not have to host the game.
The connection between users is established with peer-to-peer technology. The

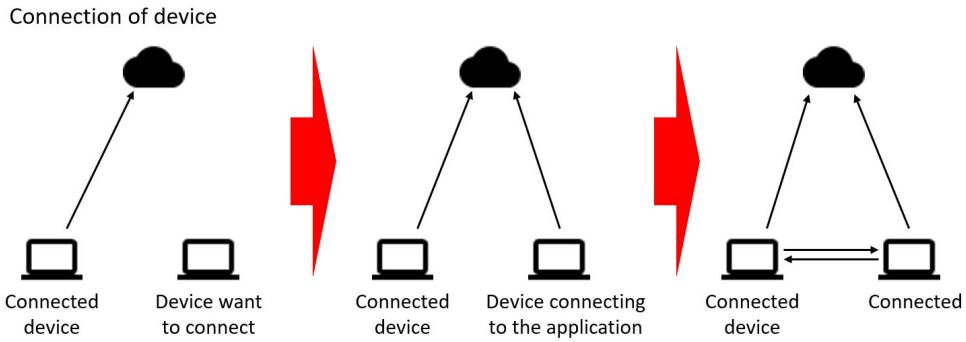


Figure 3.4: P2P connection mechanism.

discovery of devices is ensured by the server. The server will communicate and help devices to connect them in a peer-to-peer way. The peer-to-peer connections use sockets, but not classical ones. These sockets are specific sockets from libraries design to creates a peer-to-peer connection from the main server used for discovery.

Libraries connecting users in peer-to-peer way and using event-based communication to exchange information are not plenty:

- webRTC[1]: powerfull but not easy to use
- socketIO [2] and p2pSocketIO [3]: webRTC based, simple but some bugs still present.

Those propose other functionalities as event-based communication which is a cloud typical way to exchange information between client and server. Using these functionalities reduce significantly the modifications to do to adapt a cloud application.

Client Code Adaption to Host the Application

The hosting part is the most complicated to adapt. Indeed, it is highly probable that the hosting device uses the application. It must ensure the behavior of the local services and exchanging information with other clients without interruption or affecting the user experience. The code of the server part can not simply move

[1]https://webrtc.org/
[2]https://socket.io/
[3]https://socket.io/blog/socket-io-p2p/

from the server file to the client file.

Client code adaptation for a cloud application to an edge is challenging if you want to keep simple code without code duplication and reducing the changes in the code. To manage the hosting part, the client must have multiple adaptions.

First, the **server specific libraries** must be tested if they can be used in browsers and replaced in case it is not possible. For example, the libraries used to create HTTP servers are not usable in classical browsers. Clients do not have ports configured for this job and the browser is not authorized to do it. Browsers also do not support the network mechanisms of servers. If there is a library that can not be used, an alternative must be found or implemented.

Secondly, the **local behavior** of the hosting device must be modified if the device hosts the application. We must add the local reaction to server-side events broadcasted in reaction to other client messages. Also, local behavior must be adapted to fit with server events. For example, if a client sends a message which must be broadcast by a server to all clients, the hosting device which can not send a message to its own hosting part, must send directly the server broadcast event and not the client one. Clear cloud uses require to use of different event names for the event sent by the client and by the server. A hosting device will react to client-event as they were server events and will replace the client-side message with the response of the server and broadcast messages to other players if necessary. To facilitate the adaptation, specific functions can be added to the events or behaviors, only the content of this function must be adapted. This allows keeping the same code between cloud and edge versions. Figure 3.5 shows how to do it and how it impacts the code. *The migration mechanism* has to be added. Interruption of the service can happen if the old host stops his server role before the new host begins. Verification mechanisms are really important and essential for good migration. This avoids and protects against interruption, we take inspiration from virtual machine cloning. The first step is to copy the state variables and activate the new host, during few moments there are two hosts with the same state, they both received the new data. After, the new host will send a message to confirm that he activates the hosting and store the state variables. After this, the old host sends a message to finish the migration and stops hosting. This mechanism is illustrated in figure 3.6. This mechanism has just to be added. Because nothing like this exists in the cloud version. In figure 3.7, a pseudo code explains the block to add in cloud adaptation.

User or Server Disconnection

User disconnection can be problematic, indeed, if the disconnected user was hosting the whole or a part of the application, it must be replaced. Depending on the

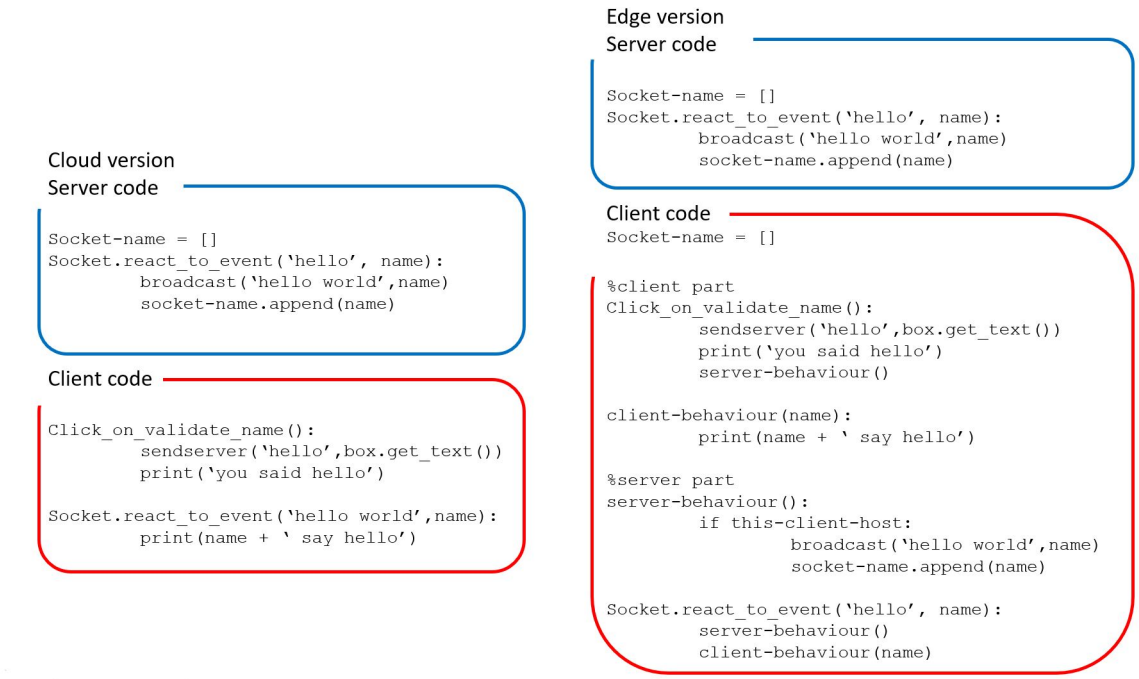


Figure 3.5: Pseudo code cloud adaption to edge. The server code is not modify for the behaviour to event reaction. The code of the client is modified. The server and the client behavior are extracted. The behavior functions are called when they received or generate the event if this client host the application.

library, only the server can detect disconnection (socketIO and p2psocketIO). If we used state save security described in paragraph 3.2.3, the state recovery is not a problem but, to avoid perceptible interruption, a new host has to be set quickly. The server will design a new host when a disconnection event happens. Depending on the type of application, the state of the service can change in case of disconnection (e.g.: a game where a leaving player broke the game). To ensure the continuity of the service, the server will designate a random new host or will host himself.

This adaptation of cloud application is easy for this protection, the mechanism must just be added and adapted depending on the application and to the state consistency and security policy.

A server disconnection or failure will disable the server function and other protections ensured by the server. It means:

- No user disconnection protection (if library used is socketIO)
- No new user connection

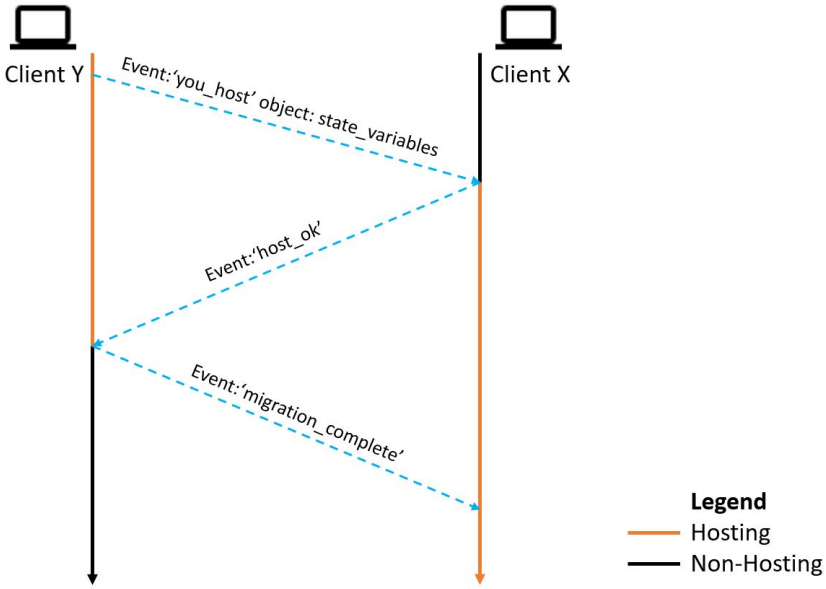


Figure 3.6: Migration mechanism between two hosts.

But these problems are the same in the cloud paradigm in case of server failure. Edge computing presents more advantages than cloud: the service will continue to work and connected users, will not notice the change. There is no adaptation needed to deal with server disconnection.

3.2.5 Separation of the User Interface

Leveraging the low latency of edge computing, a separation of the interface takes advantage of the device characteristics diversity. Edge device context is composed of a high diversity of device types. Different characteristics can be highlighted. A touchscreen, screen size, a keyboard, and many more features exist. For example, we will not display a complex picture on the small screen of a smartwatch or using an interface needing high reactivity of the user on a tv using a simple remote control.

The separation of the user interface consists to split the interface into multiple blocks which can be moved from a device to another. These blocks contain the display of the interface concerned and the functions linked. A block can also be a specific functionality of the application. For example, the modification of a block. A visual part can propose two different blocks.

The easy way to decompose GUI is to separate the visual blocks and make them independent of the device. Create roles for device and activating or not blocks of

```
State_variables;

%part to delagate the hosting part
function migrate(new_host_id):
        send('you host', {« dest »:new_host_id, « state »: state_variables})

Socket.on('host ok'):
        hosting = false
        send('migration complete')

%part to receive the hosting part
%myId is an unique id of each clients, the id is based on the socketID
Socket.on('you host',{« dest »:id, « state »:state):
        if id == myId:
                hosting=true
                state_variables = state
                send('host ok')
```

Figure 3.7: Pseudo code example added for host migration adaptation. The event exchange start with the migrate function. When the new designated host receives the `'you host'` event, he starts to host, register the states and send back a `'host ok'` to notify the old host that he is hosting the game. The old host will stop hosting when he receives this last event.

the GUI depending of the device role.

3.3 Conclusion

We design our solution of edge computing to adapt cloud applications to edge applications. Multiple types of applications can be converted. This conceptual framework was designed in sight of developing a proof of concept. This solution can work without a server except for device discovery and user disconnection protection.

Code modifications are multiples. Behavior adaption and adding hosting mechanism. Disconnection and state management are modules that can be added if the needs of the application require them.

Chapter 4

Proof of Concept

In this section, we will apply the conceptual framework and the adaption methods explained in the previous chapters 2 and 3 on a concrete application. We choose a legacy application onto which we apply our methodology. We will modify this cloud application into an edge one. We will also adapt the graphical user interface to detach some parts so it can adapt to the available devices. All examples and code implementation can be found in the appendix A.

4.1 Cloud and Edge Application Structure

In this section, we will examine the cloud application choose and how it works. We will also describe the shape of the edge application after the adaptation.

4.1.1 Cloud Application

The choice of the application presents multiple requirements. The application must be a simple cloud working service. It must have low latency requirements and require to interact with edge devices. An interactive graphical game is ideal and fits the requirements. The type chosen is a Pictionary game. This allows the implementation of a coherent graphical user interface detaching and leverages device diversity to split the user interface. The game is multiplayer, the server must host multiple users and types of devices. This game can be used in concrete scenarios. This is a good application for a proof of concept for our edge computing adaptation [24].

The application must respect multiple requirements. First, it has to be a simple open-source project. After, it must fill important technical requirements to have a representative cloud application. We searched for a working cloud Pictionary based on common technologies to have a good delegate of classical cloud applications.

We found a Pictionary game using nodejs. We tested it and except for some small bugs, the game was working.

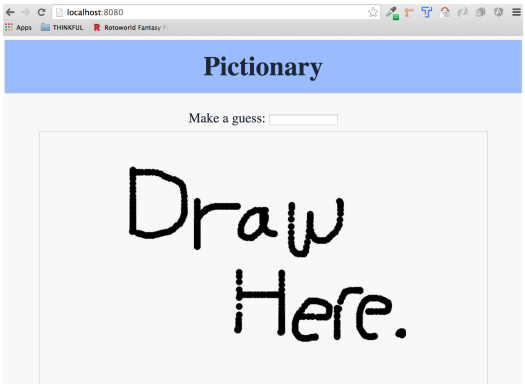
Pictionary is a good application because the game requirements fit with an interactive multiplayer application. The application must be usable in real life by every user wanting to play. The game is inspired by Pictionary, but some rules were simplified.

The traditional rules implies a game board (Figure 4.1) with colored cell corresponding to categories of draw [25]. Pawns are moved to different cases following the instruction of a dice. Players are split into a maximum of four teams. To move, they have to guess the word drawn by their partner(s).

Figure 4.1: Traditional pictionary board game. [26]

There are modifications of the rules of the Pictionary application compared to the original game. The Pictionary application does not use a board or draw category. It simply picks a random word in a dictionary. Players are clients or the application. They can have two roles: guesser or drawer. Guessers see the draw, enter a word guess proposition and see the last guess made by other players (Figure 4.2b). The drawer makes a drawing representative of the word that other players have to guess (Figure 4.2a).

(a) Drawer graphical user interface of the unmodified cloud pictionnary [24]



(b) Guesser graphical user interface of the unmodified cloud pictionnary [24]

Figure 4.2: Graphical user interface of the unmodified basic application.

Preparation of the Application

Before adapting the application, we notice and patch some major bugs. We correct the code to avoid a fatal server error when a player disconnects. We block the drawing ability on the canvas when a player has the guesser role. We set a reset of the canvas when the word is guessed.

To fit concrete scenarios, we added features that were required to fit our scenario. Before adapting the computing paradigm of the application, we add five features. The first was a score count to display the score of each player. The second was a winning mechanism. When a player reaches a score of X points, the game restarts and displays the winner. The third is a waiting feature when only one player is connected. The wait is displayed until other players join the game. The fourth is a saving of the drawing of the current game. This server will send the actual drawing to every new player joining the current game. The last added feature is the ability to support drawing with a touchscreen.

The score count was implemented only on the client-side. If the device reloads the page, the score is lost. This will not influence the adaption to edge computing.

Structure of the Application

The Pictionary application is a classical cloud application. The application is composed of multiple parts. We will review all these mechanisms before the adaption. We will adapt these mechanisms, it is important to keep in mind how they work to understand the operation needed for the adaption.

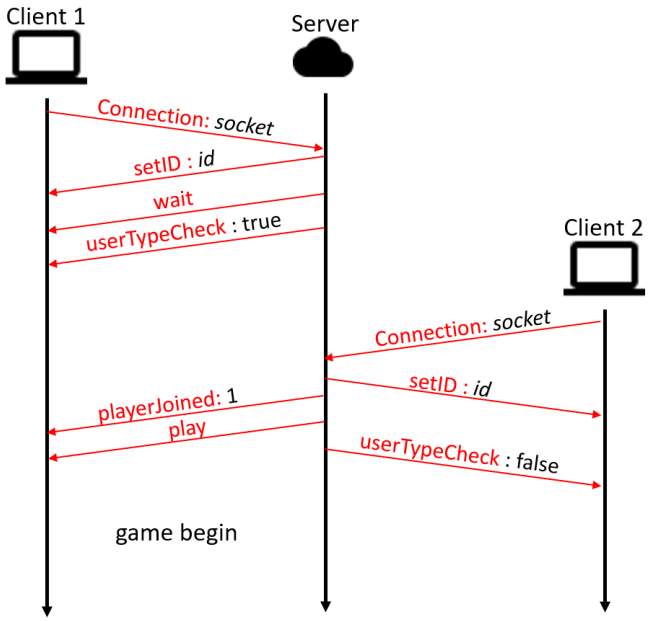


Figure 4.3: Cloud event connection exchange.

The connection mechanism is quite simple. From the user point of view, he connects to the Pictionary and the game begins when two or more players are connected. Cloud connection gives to each player a role and makes the player waits for other players if it is first connected. The cloud connection event exchange can be seen in figure 4.3. The event named `'connection'` is a native socket.io event that is sent when a client connects. Connection events provide the socket address of the device that can then be used to send message. This event is only generated with the server library.

The event `'wait'` is sent if the user is the first connected and must wait for other players to begin to play. The event `'play'` is broadcast if the user is not the first connected to start the game. After that, the event `userTypeCheck` assigns a role for the game to the player.

The drawing mechanism is a simple exchange of implemented events. The drawer draws the circle and sends the position to the server with an event named: `'clientToServer'`. The server then broadcasts the position to all the players so that they can display the new circle.
Figure 4.4 shows the drawing events exchange in the cloud paradigm. The host registers all points in case of connection of a new player. If a new player connects, the server will send all the points of the actual drawing, so the new player can display the same drawing as the other players of the actual game.

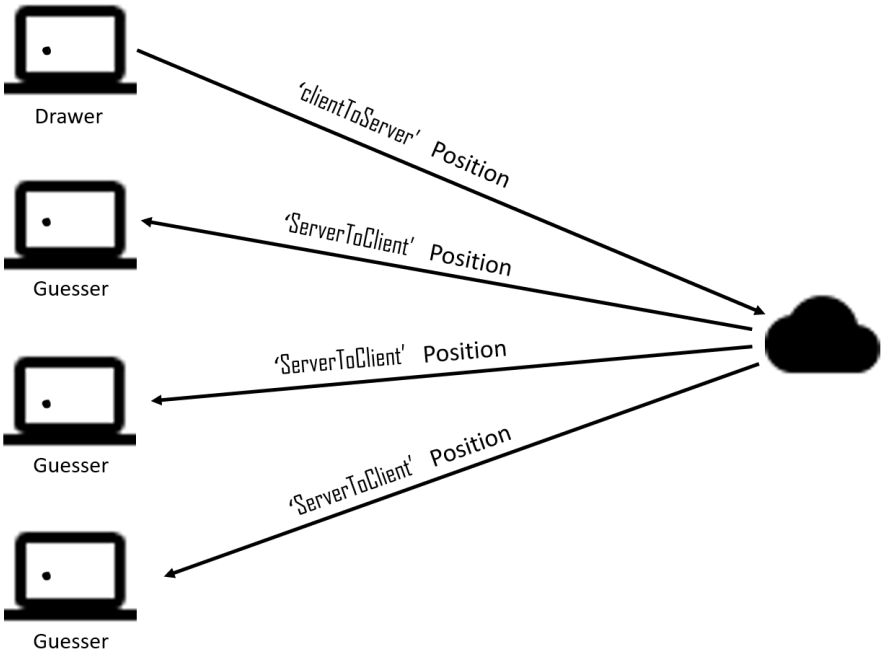


Figure 4.4: Drawing event exchange.

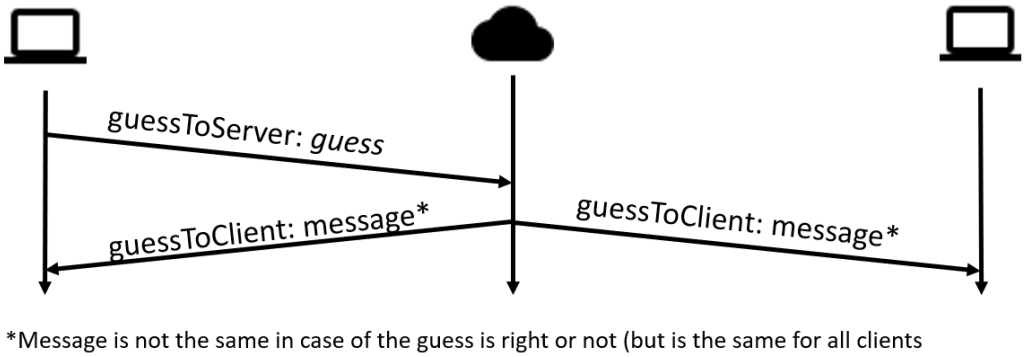


Figure 4.5: Guessing mechanism.

The guessing mechanism is also a typical example of simple interactivity between

the server and users. The guessing event exchange is described in figure 4.5. The word sent by the guesser is compared by the host to the word to guess. The server will send a message to notify other users that someone tried to guess the word. If the guess is right, the dictionary will advertise that the word to guess has changed and someone has found the word. The reaction of the users will be to reset their role and their canvas. The guesser will recognize that he made the right proposition and become the drawer. The drawer will reset himself as a guesser.
If the guess is incorrect, the message content is just the wrong word to advertise it to all the other players. And the game continues with no other change.

The disconnection mechanism has an event exchange a little bit more complicated. It has to replace the drawer if it is disconnected. In case of disconnection of a guesser, the server must just advertise to others users.

In figure 4.6, there is an example of cloud version disconnection of the drawer. A device disconnected sends an event "disconnect" to the server. He will advertise to other players the number of users still in the game and ask every player if they are drawers with a `'serverToClientDrawerCheck'` event. Each client will respond by giving his role with a `'drawerHere'` event. If a client is the drawer, he will attach a **`true`** value to the event, otherwise, he will attach **`false`** value. If there is no drawer still active, the server will pick a new drawer and advertise him. Then, the server will order all players to reset the actual draw.

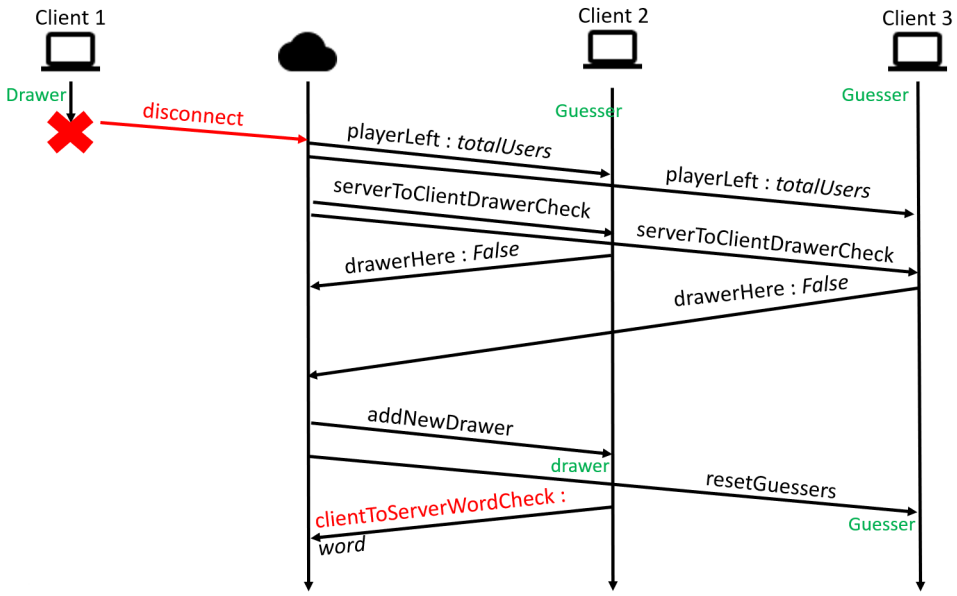


Figure 4.6: Disconnection event mechanism.

Used Libraries

The libraries used in the cloud application are important because, for the adaptation, we will need to test them and evaluate if they must be replaced or not.
http is a basic library for the HTTP server. This library can not be used on a browser. It will only be used on the server.
express is another basic library to create and manage servers. As the HTTP library, it is not usable on a browser. We will also only use it on the server.
Socket.io is the library used to create a socket to communicate between the server and client. Socket.io is a bidirectional two ways even based communication. Socket.io is used to exchange information between the server and client. Communicating in the two directions is important for stateful applications. Sockets mechanisms are adapted for this kind of communication.
SocketIO is used in other multiple libraries as socket.io-P2P.

These libraries are the three libraries used in the application open source before any modification.

4.1.2 Edge

The goal of the adaptation is to have an edge computing application. The edge application will respect the description made in the paragraph 3.1.2. In this version, all the server game mechanisms are adapted to be host on the user device. The connection mechanism, the guessing mechanism, the drawing mechanism, and the disconnection mechanism have all to be adapted to keep the same behavior from the user point of view. We will add a migration mechanism as described in subsection 2.1.2. We will also propose a separation of the user interface into three device roles: a player device, a drawing device, and a display device.

Libraries

To accomplish this, we have to test and replace some libraries used in the cloud version. Socket.io-P2P is added in the client code. Socket.io-p2p-server and ecstatic are server libraries used for edge computing.
Socket.io-P2P is the library used to create peer-to-peer sockets used for communication between clients.
Ecstatic is an npm library use to replace express for compatibility with socket.io-p2p purpose.
Socket.io-p2p-server is the library use on the server to help the connection between peer-to-peer sockets.

4.2 Cloud Computing Application Adaptation to Edge Computing

We have exposed the chosen application and its structure. We will see all the modifications we have to do to adapt the cloud application mechanisms. We will apply the adaption technics explained in the chapter 3 to adapt the connection mechanism, the drawing mechanism, the guessing mechanism, and the disconnection of server or user mechanism.

4.2.1 Adaptation of Existing Cloud Mechanisms

The mechanisms of cloud applications must be adapted to be hosted by the user device. The hosting application must manage its behavior and ensure interactivity between all the players.

Adaption of the connection mechanism is important to keep the same behavior. The server has to delegate the hosting of the application to the user's device. Except for the play and wait event, the initialization is fully executed only by the server. But in the edge version, the server has to help to connect users with peer-to-peer. Figure 4.7 shows an example of the connection protocol of a new player to the p2p network. The event `'go-private'` is sent by the new client to the peer-to-peer host when he establishes a p2p connection and makes the sentence 'waiting for p2p connection' disappear. The go-private message asks the host the drawing representing the actual game for the new player to allow him to play with other players.

After this message, the new player is connected with peer-to-peer to the user device host and plays the game.

If the device is the only one connected, the server will give him the hosting role. The state of the application is set by default and the server will just ask to activate the hosting but no states have to be transferred this time. The device will wait until others players connect.

Adaptation of the drawing mechanism is a simple concrete application of the code adaption of events exchange described in section 3.2.4.

Adaptation of the guessing mechanism has to be done in the same manner as described in section 3.2.4. The code of guessing being a little bit more complex than the example in pseudo-code, the adaption needs more attention. However, this is a complex example of the conceptual adaption of chapter 4. The complexity of the adaptation is due to the different ways to react to the same event.

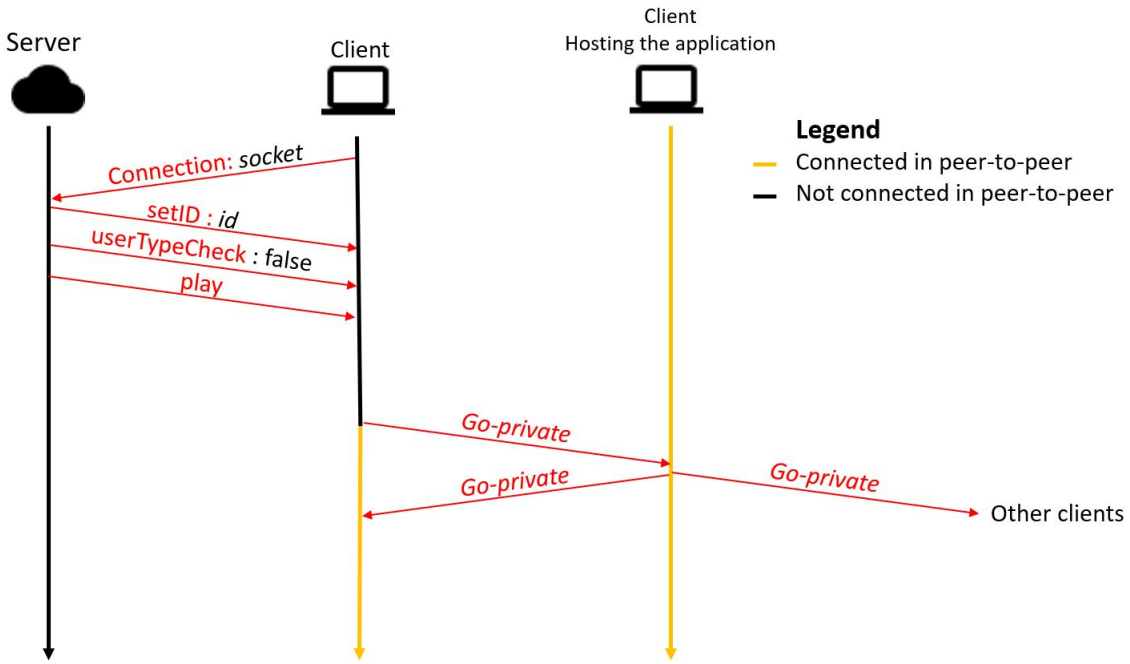


Figure 4.7: Edge connection event exchange.

Adaptation of the user disconnection mechanism need more modifications than a simple adaptation of event exchange. The edge device is not able to detect a device disconnection with the `socket.io-p2p` library. In the cloud version, the disconnection of a user is detected by the server. In the edge version, the user devices are not able to detect a disconnection. The server will detect it and advertise the players.

The server will first check if the disconnected device was hosting the game. If it's the case, it will choose another device to host. After this verification, the server will send to the host a message to order a check if there is still a drawer connected and the device hosting the game will reproduce the cloud disconnection mechanism as described in paragraph 4.1.1.

4.2.2 Adding Special Edge Mechanism

Now we have adapted the mechanism of the Pictionary game himself, we have to add the specific edge mechanism. The Pictionary application being state loss tolerant, we will not implement protection against state loss. We will add only one mechanism, the migration mechanism.

The hosting part is not split. The application uses two types of migration. A

starting migration from the server to the edge device and a live migration between two edge devices. These two types of migration are explained in the paragraph 2.1.2.

The states of the application are only saved on the host device. We use, therefore, the transfer mechanism described in subsection 2.1.2. By sending first the state and starting the hosting at the same time. The states of the app are the save of the actual drawing and to word the drawer is trying to make guess.

4.2.3 Edge Additional Protections

In this section, we will explain two protections we have to add to ensure smooth running. The first is the security to send events to the right receiver. The second is the idempotency.

Sending to the right host is a little bit more complicated than cloud computing. In a peer-to-peer network, the devices are all connected and they send to all devices. Messages are sent to all users and the structure must be adapted. For all messages which are not broadcast, the receiver must be indicated with a unique identifier. In our application, we will use the id of the socket as the device id. The variable or object joined with the event is adapted with this id. The first case is if there are no variables that join with the event, the id will be placed as the variable. And if there is already data send with the event, we will create a dictionary with two fields, one with the id ("ar") and another with the object (the name can change to make more sense). The server must also be adapted to use the same functions as the client code. Figure 4.8 show an generic adaptation for this last use case. *Idempotency* mechanism is needed because socket.io-p2p library can send multiple times the same message. But this can modify the behavior, so we add a protection mechanism where multiple times the same message can modify the state or the behavior. The protection result is that applying multiple times the same message is the same as applying it only once, which is useful for at-least-once delivery models. We use an incremental message id to avoid applying two times the same event content. Each user will keep in memory the last received message-ids and will only apply the received message if the message-id is unregistered.

4.3 Graphical User Interface Separation

Now we have implemented an operational application, we will propose the user interface separation. The interface design is composed of two elements:

- The role of the device (the guess box, or the word to guess)

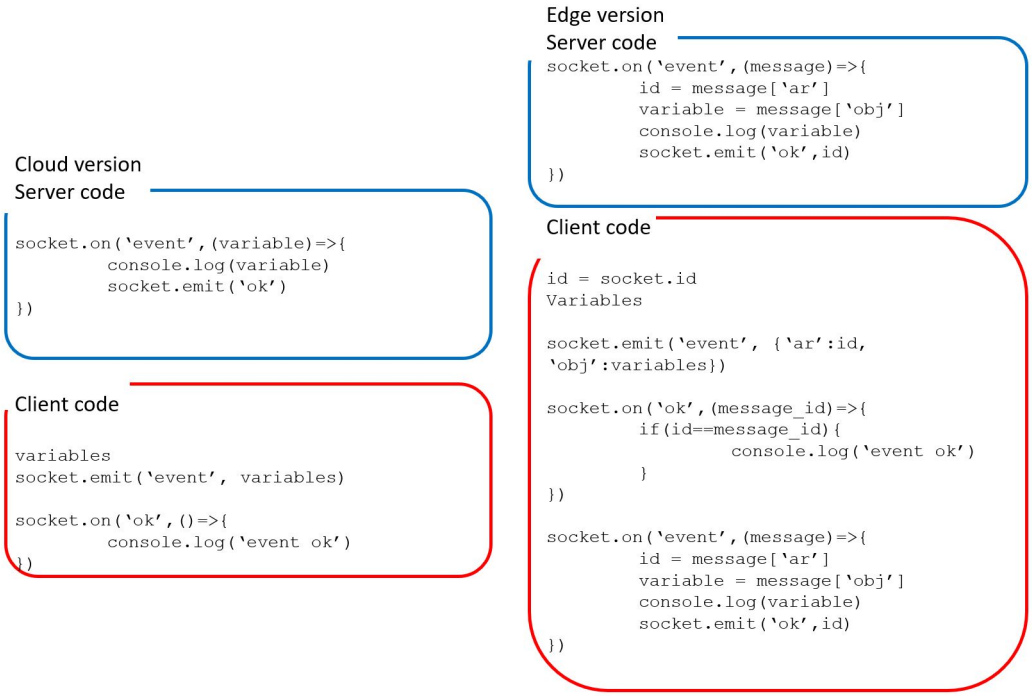


Figure 4.8: Example of code adaptation to send to the right receiver.

- The canvas to draw and display the draw

Functionalities or mechanisms can be linked to elements of the graphical user interface. We can split the interface and the functionalities linked to it on multiple devices. We add options and propose two scenarios. They are adapted in two versions, cloud and edge computing. Options allow to organize different blocks of the interface. There are two propositions of possible scenarios.

4.3.1 Non-shared Scenario Interface

This is the basic scenario proposed by the cloud, all the functionalities are grouped on the device of the client. It's the most simple way to play. The interface is not shared between multiple devices. No more modifications are needed. An illustration of the repartition of the functions of the interface can be found at the left of figure 4.9.

The better way to understand the scenario is to have an example. Imagine a family of five people. The father named John, the mother Sierra, three kids Riri, Fifi, and Loulou. The dog is called Cannelle but she will not play the Pictionary application. In this first scenario, all the members of this family are in different

rooms. They all use a personal device (smartphone, pc, tablet). Each device displays the drawing, the word for the drawer or the guess box for the guessers, and the drawer draw on his device. They can be in the same room or all around the city and play the game.

4.3.2 Shared Scenario Interface

This scenario is favorable when players are all in the same room. It leverages the low latency of the edge computing paradigm to split the user interface into three different interface functions: the user's role, drawing device, and the display drawing device. The user's role display only the drawer or guesser area. This area display the word or the guess box to enter a proposition of guess link to these roles.

The second function is the drawing device: a touchscreen device, or a device to draw easily. A tablet or touchscreen smart television is the best for this role.

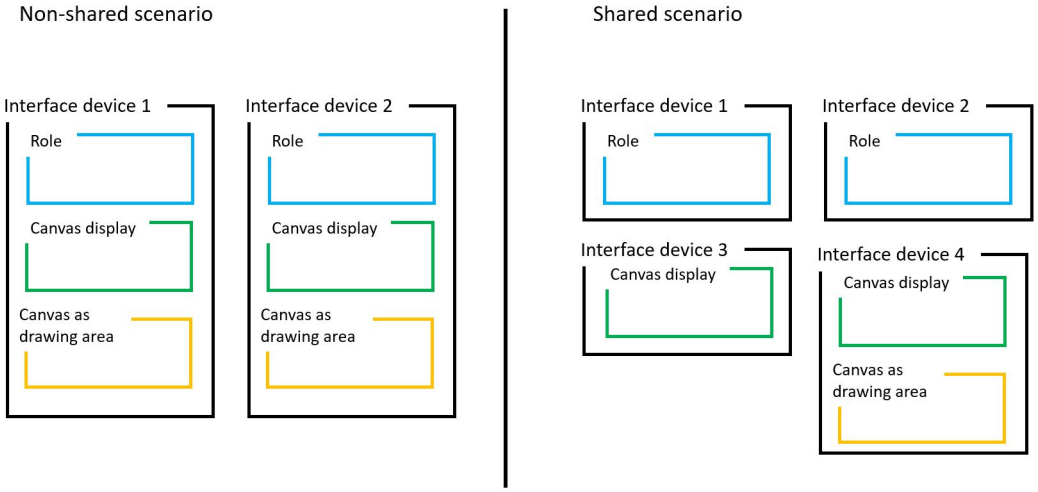


Figure 4.9: Different scenarios with role distribution.

The third function is the display device. This device will display the drawing for all users. An illustration of the repartition of the function for this scenario can be found at the right of the figure 4.9

Our family will play this version with user interface separation. John starts his device and connects the smart television to the game. He sets the TV to display the drawing. Riri starts also his device and connects the tablet to the games. He sets the tablet to draw function and passes the tablet to Loulou which is the first drawer. All personal devices are set to display only the word to draw or the guess

box depending on if the user is a drawer or a guesser. They will see the drawing on the tv. Sierra guessed the word drawn by Loulou. Loulou gives him the tablet to let her draw the new word display on his device.

4.4 Conclusion

In this chapter, we have seen an example of the adaption of a cloud computing paradigm-based application to an edge computing paradigm based. We also improve the application by adapting the interface. The code of the two versions of the application can be found in the appendix A. The next chapter will describe the evaluation protocol of the applications and summarize its results.

Chapter 5

Evaluation

In the previous chapter, two different versions of the Pictionary application were developed, i.e., a cloud computing-based version and an edge computing-based version, both supporting the non-shared scenario and the shared scenario, which is assumed to leverage the low latency and diversity of edge devices. In this chapter, we evaluate these two versions with their respective scenarios to determine their potential effect on usability and latency perceived by the end user.

5.1 Experiment

To determine the similarities and differences between the cloud computing-based version and the edge computing-based version, a controlled experiment was conducted based on two conditions: a control condition represented by the cloud computing-based version and a new treatment represented by the edge computing-based version. We made the following hypotheses:

H_{11} = *The usability of the Pictionary application developed for the experiment is similar to the usability of average systems.* The end user does not notice any significant usability difference between the Pictionary application developed for the purpose of testing cloud vs. edge computing with respect to interactive applications with average usability.

H_{21} = *The latency perceived by end users in the edge computing-based version is better than the one perceived in the cloud computing-based version.* In the shared scenario of the Pictionary, when there is a wide diversity of devices in terms of system response time, the edge computing-based version offers a superior perceived latency than the cloud computing-based version.

5.1.1 Participants

By word of mouth, we recruited 10 participants (2 females, 8 males) aged between 22 and 60 years (M=30.3, SD=13.9) coming from various disciplines and backgrounds: 3 from civil engineering, 2 from computer science, 1 from linguistics, 1 from marketing, 1 from economic sciences, 1 from political sciences, and 1 from history of art. Only one participant did not know the game of Pictionary beforehand while four participants regularly play the game. All participants declared using a daily computer or a smartphone, but less frequently a tablet, except one participant.

5.1.2 Apparatus

Nowadays, the computational power and capabilities of portable devices like smartphones and tablets are improving so much that they almost match those of desktop computers. However, this is not always the case: device diversity prevails when various end users come with their own devices, exhibiting a wide range of computational power. Device diversity and portability are exactly two factors that are specifically addressed in edge computing. Therefore, to reflect this diversity, we tested several devices brought or mentioned by the participants in three categories, i.e., a smartphone, a tablet, and a desktop computer, to select three representative cases characterized in Table 5.1.

Device category	Desktop	Tablet	Smartphone
Device	Alienware M15	Apple iPad pro	Xiaomi MI9t
Year of production	2015	2020	2019
Pen based	No	Yes	No
CPU	Intel I7-4720HQ	M1	Qualcomm Adreno 618
Amount of cores	4	8	8
Frequency	2.6 GHz	3.2 GHz	2.2 GHz
Screen resolution	1920 × 1080	2732 × 2048	2340 ×1080
Touchscreen	No	Yes	Yes
Screen size	15.4 inch	12.5 inch	6.39 inch
Keyboard	Yes	No	No

Table 5.1: Characterization of devices selected.

These devices were tested to measure their device latency, defined as the time elapsed between the drawer starts creating a new symbol and its display on a guesser canvas of the device. We measured this latency for each device hosting the server role. Without any delay added to the edge version, it is difficult to notice

Delay added [ms]	Device latency [ms]
0	8
100	120
150	170

Table 5.2: Latency measured depending on the delay added.

any difference between the two versions since our application is light to host on the three devices selected. The average latency is about a few milliseconds on all devices due to the performance of the application. Adding a delay to the cloud version with a traffic controller results in a latency ranging from 8 ms to 20 ms (Table 5.2). Consequently, we defined 2 scenarios (S=shared vs. N=non-shared) × four possible configurations (giving 8 conditions: see Table 5.3) as follows:

- Cloud computing-based version with no added delay (S0/N0): this configuration represents an ideal cloud-based interactive application with a user connected directly to the server. This configuration is used as a baseline.
- Cloud computing-based version with an added delay of 80 ms (S80/N80): this configuration represents a cloud-based application with good latency.
- Cloud computing-based version with an added delay of 300 ms (S300/N300): this configuration represents an application with a latency for a server far away from the end user.
- Edge computing-based version with no added delay (SE/NE): this configuration represents a typical usage of edge computing.

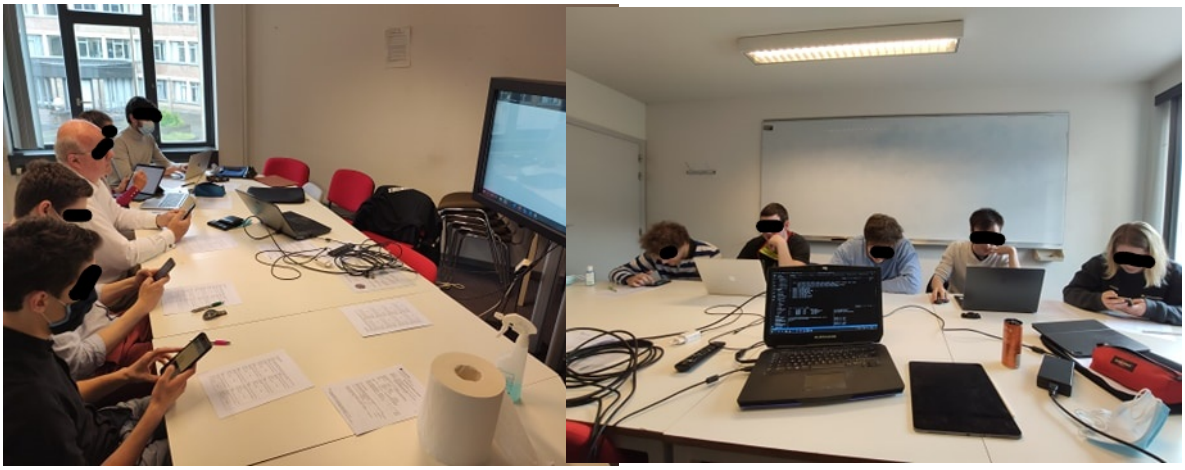
	Cloud 0 ms	Cloud 80 ms	Cloud 300 ms	Edge 0 ms
Shared scenario	S0	S80	S300	SE
Non-shared scenario	N0	N80	N300	NE

Table 5.3: The eight conditions of the experiment with their code.

5.1.3 Task and Stimuli

The experiment took place in two separate sessions of about 25 minutes, one session for each group of five members, a constraint induced by the sanitary crisis at the time of the experiment. Each group was invited to play the Pictionary game with the following rules. Each game is played until a player wins. To minimize impatience due to long games, the winning score is set to 2 points. A list of simple

words to draw was prepared and randomly sorted. As per stimuli, we selected simple words which do not require any special understanding or drawing capabilities: car, flower, house, cloud, tree, light bulb, mobile phone, shoes, dishes, foot, balloon, beach, candy, gun, plane, chair, table, shower, bed, money, mouse, knife, boat, heart, square, hand, bird, sheep, elephant, fork, eyes, banana, cow, hammer, turtle, arm, cd, motorbike, lollipop, and sun.



(a) First group of the experiment. (b) Second group of the experiment.

Figure 5.1: Setup for the two groups of five participants.

5.1.4 Setup and Procedure

Before the experiment, participants were introduced to the procedure and informed that they could leave the experiment at any time. They were then invited to sign a GDPR-compliant consent form and filled in a demographic questionnaire on their age, highest education level, current occupation, etc. Participants were then invited to enter a quiet meeting room and to take a seat in front of a table where to place their device flat on the desk. To preserve the ecological validity of the experiment, participants were allowed to bring and use their own device: 3 different smartphones and 2 computers per group. They were then aligned along a table in front of a large vertical display (Fig. 5.1).

The course of the full experiment was explained and it was only mentioned that the participants will evaluate different versions of the Pictionary game through an application described to them. Participants were then given a few minutes to try the application out and look at the various screens before continuing the experiment. The rest was divided into five tasks:

1. *Configuration*: the participants configure the application for their device.

2. *Game*: the participants play 8 games, i.e., four in the shared scenario and four in the non-shared scenario, corresponding to the eight conditions summarized in Table 5.3. They were randomly assigned without participants being notified about which condition was running.
3. *SUS Questionnaire*: after the games, the participants were asked to fill out the System Usability Scale (SUS) questionnaire [27], which comprises ten questions evaluating the usability of a system (in this case, the Pictionary application). Each question is stated on a 5-point Likert scale (1=strongly disagree to 5=strongly agree). Five questions are positively stated while the five others are negatively stated in order to avoid repeating entries. This questionnaire was selected for its proven reliability [28], for its high correlation with usability ($\alpha = 0.92$) [29], and for its efficient administration.
4. *Open-Ended Questions, perceived latency, and comments*: after each game, the participants answered two open-ended questions (What are the strengths of this version and what are the aspects that need some improvement?) to provide insights on their overall feeling, free-form comment, and the perceived latency of the version (what is the latency of the application on a 7-point rating scale, 1=very small to 7=very long).
5. *Ranking and Criteria*: after completing all games, participants were asked to rank the versions in decreasing order of perceived usability (from 1=the most usable to 8=the least usable) and to state the three most favorable criteria (ranked from 1=the most important to 3=the least important) and the three least favorable ones.

5.1.5 Quantitative Measurement

While time was not a constraint during each game, participants were timed and their actions were recorded in a log file with a timestamp, along with the real system latency computed for each significant action. In this way, it is expected to compare the latency perceived by participants with the real latency computed by the system.

5.1.6 Design

Our study was therefore a within-factor design (since all participants evaluated the eight conditions) with one independent variable representing the eight conditions (Table 5.3), and with five dependent variables:

1. SUS Question score: integer variable with 5 values representing the individual score assigned by participants to each SUS question.

2. SUS Overall score: real variable representing the overall SUS score computed for all questions [29].
3. Perceived latency: integer variable with 7 values representing the latency perceived by participants (1=very slow, 7=very fast).
4. Real latency: real variable representing the average system latency computed at run-time for each participant.
5. Game ranking: numerical variable with 8 values representing the subjective order of usability perceived by each participant for each game.

5.2 Results and Discussion

5.2.1 Effect on Usability

We computed d'Agostino-Pearson and Anderson-Darling statistical tests to determine whether the questions follow a normal distribution. Since at least one test failed for each question, we computed one sample Wilcoxon signed-ranked test to determine whether the average answer for each question is statistically significantly above (in case of a positive statement) or below (in case of a negative statement) the median value of 3. The distribution of answers is reproduced in Fig. 5.2.

The question Q1="I think that I would like to use this system frequently" (M=2.44, SD=1.13) is below the median value of 3, which suggests that participants are less eager to use this application in a frequent way, probably because of the distributed setup.

The question Q2="I found the system unnecessarily complex" (M=1.44, SD=1.01) received a score significantly below 3 ($W = -42$, p^*=.011), which suggests that participants did not perceived the application as a complex one to use and this observation was uniform (one one participant out of ten scored '4'). This is confirmed by Q3="I thought the system was easy to use" (M=4.00, SD=0.71), which is also significantly above the median value (W=28, p^*=.015).

The question Q4="I think that I would need the support of a technical person to be able to use this system" (M=3.33, SD=1.88), altough being averaged above 3, is not significantly above this value (W=6, p=.73, *n.s.*). Similarly, the question Q5="I found the various functions in this system were well integrated" (M=3.44, SD=1.01) is not significantly superior to the median (W=14, p=.35, *n.s.*). The question Q6="I thought there was too much inconsistency in this system" (M=2.33, SD=1.12) is not significantly inferior to the median ($W = -12$, p=.19, *n.s.*). Yet, this suggests that the various GUI components were perceived consistency throughout the experiment. This is important since consistency belongs to the most important usability factors. The question Q7="I would imagine that most people would learn to use this system very quickly" (M=4.22, SD=1.09)

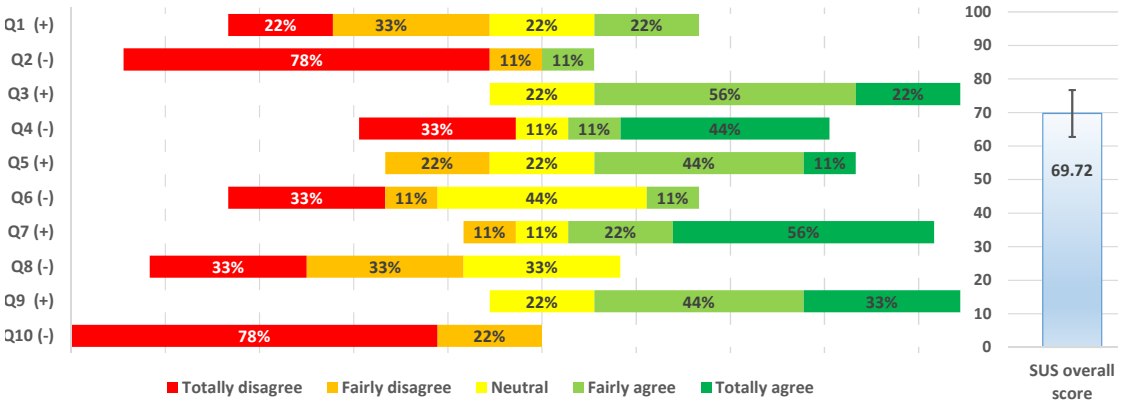


Figure 5.2: Distribution of participants' answers to SUS questions and SUS overall score. (+) depicts a positive statement and (-) depicts a negative statement. Error bars show a confidence interval of 95%.

is significantly superior to the median (W=32, p^*=.031). This question received the highest average among all questions, thus suggesting that learnability of the system was the mostly appreciated. This is confirmed by question Q8="I found the system very cumbersome to use" (M=2.00, SD=0.86) is significantly below the median ($W = -21$, p^*=.031). The question Q9="I felt very confident using the system" (M=4.11, SD=0.78) is also above the median (W=28, p^*=.016). The question Q10="I needed to learn a lot of things before I could get going with this system" (M=1.22, SD=0.44) is highly significantly below the median ($W = -45$, p^{**}=.003). This high correlation also reinforces Q7 about learnability, probably the most appreciated usability factor.

The SUS overall score averaged on all participants is 69.72% (SD=10.70), which represents a good score that is just above the threshold of 68% (a SUS score above a 68 would be considered above the average score of the 500 studies and anything below 68 is below average [29]). Guttman's $\lambda - 2 = 0.92$, which represents a very high score confirming the reliability of the answers. Kendall's coefficient of concordance (W=0.31, $p^{**} = .003$) indicates a fair agreement by participants when appraising the conditions.

In conclusion, the hypothesis H_{11} is considered supported.

5.2.2 Effect on Perceived vs. Real Latency

The mean and standard deviation of latency perceived by participants vs. real latency measured by the system are reported in Table 5.4 in decreasing order of the averaged perceived latency. The condition which received the longest perceived latency is the shared cloud version with a delay of 80 msec (S80: M=5.90, SD=1.10),

Condition	Perceived latency					Real latency		Any difference (KW)?		
	M	*SD*	*W*	*p*	Sig.?	*M*	*SD*	*H*	*p*	Sig.?
S80	5.90	1.10	53	0.0039	**	138.42	81.59	707.8	≤ 0.0001	****
S0	5.56	1.42	32	0.031	*	60.86	52.38	441.1	$= 0.066$	*n.s.*
N0	4.90	1.97	19	0.24	*n.s.*	64.14	38.30	487.3	$= 0.0068$	**
S300	4.90	1.73	30	0.12	*n.s.*	439.13	303.09	1031	≤ 0.0001	****
N80	4.67	1.50	9	0.31	*n.s.*	117.01	42.54	716.0	≤ 0.0001	****
N300	4.30	1.16	10	0.44	*n.s.*	504.61	543.70	991.2	≤ 0.0001	****
NE	4.22	2.17	5	0.79	*n.s.*	12.25	9.41	155.0	≥ 0.99	*n.s.*
SE	3.40	1.52	3	0.75	*n.s.*	30.11	27.77	347.6	≥ 0.99	*n.s.*

Table 5.4: Correlation between perceived latency and real latency. Conditions are sorted in decreasing order of their averaged perceived latency (M=mean, SD=standard deviation, W=Wilcoxon coefficient, H=Kruskall-Wallis value, p=p value, Sig?=significance).

followed by the shared cloud version without any delay (S0: M=5.56, SD=1.42). Both were perceived as highly significantly higher (S80: W=53, $p^{**} = .0039$), resp. significantly higher (S0: W=32, $p[^{**} = .031$), than the median value 4 based on one sample Wilcoxon signed-ranked test since the distribution was not normal. All other conditions were not significantly departing from the median value (NO, S300, N80, N300, NE, and SE: *n.s.*). Surprisingly, the two worst conditions with a delay of 300 ms, i.e., S300 and N300, were not perceived as such since they appear after less constrained conditions (S80, S0, and N0). The two last conditions are the two edge conditions in the non-shared scenario (NE: M=4.22, SD=2.17) and in the shared scenario (SE: M=3.40, SD=1.52), respectively.

In conclusion, these two last results, in contrast to all others, suggest that the two edge-based conditions have a perceived latency that is better than in the cloud computing-based versions, thus supporting the hypothesis H_{21}.

To reinforce the position of the edge computing-based conditions and to determine whether there is any significant difference between the edge-based versions and the cloud-based versions, we computed a series of Kruskall-Wallis statistical tests with Dunn's multiple comparisons (n=8, α=0.05). Only one significant difference was found between SE and S80 (MR=15.50, p^*=.0203), as depicted in the left part of Fig. 5.3. In this figure, error bars represent a confidence interval of 95% (α=.05), not the standard deviations as reported in Table 5.4.

The middle large column of Table 5.4 shows the average real latency measured by the system throughout the experiment corresponding to the conditions sorted in decreasing order of their perceived latency. As expected, the two slowest conditions are those with a an additional delay of 300 msec., i.e., N300 and

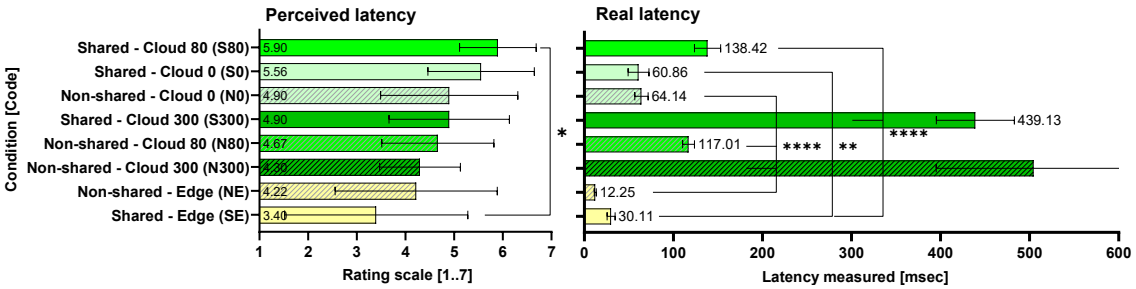


Figure 5.3: Perceived latency (left) vs. Real latency (right). Error bars show a confidence interval of 95%. Significance level is indicated as follows: *= $p \leq .05$, **= $p \leq .01$, ****= $p \leq .0001$.

S300, respectively. Apparently, the non-shared version N300 was slower than its shared counterpart S300. Then, various conditions all come as related pairs: S80 (M=138.42, SD=81.59) and N80 (M=117.01, SD=42.54), S0 (M=60.86, SD=52.38) and N0 (M=64.14, SD=38.30), and finally the edge versions SE (M=30.11, SD=27.77) and NE (M=12.25, SD=9.41). We observe that the fastest version is the non-shared edge version, which benefits from the lowest averaged latency (M=12.25), but also the narrowest standard deviation (SD=9.41) among all conditions, which reinforces the position of this edge version as the first one.

Similarly to the perceived latency, we computed a series of Kruskall-Wallis statistical tests with multiple comparisons (n=8, α=0.05). Since only three comparisons were not statistically different in all 28 possible comparisons, we reported significant differences in the right part of Fig. 5.3 by scenario only, i.e., within shared and within non-shared, and with the edge version as a baseline, to preserve the legibility of the figure. In the shared scenario, the SE condition is highly statistically inferior to SO (MR=71.06, $p^{**} = .0058$), and very highly inferior to S80 (MR=192.02, $p^{****} \leq .0001$) and S300 (MR=347.09, $p^{****} \leq .0001$). In the non-shared scenario, the NE condition is always very highly statistically ($p^{****} \leq .0001$) inferior to N0 (MR=193.4), N80 (MR=321.9), and N300 (MR=450.3). These statistical results confirm the prescribed order of conditions depending on the scenario and the added delay, if any: the scenarios with a delay of 300 msec are slower than those with a delay of 80 msec, without any delay, and in the edge-based version.

To determine whether there is any effect of the scenario on latency, we further computed a series of Kruskall-Wallis tests with Dunn's multiple comparisons by pairs (each shared version vs. its corresponding non-shared version). With respect to the real latency, only the non-shared version NE is significantly better than its shared counterpart SE (MR=155.7, $p^{****} \leq .0001$). With respect to the perceived latency, no significant difference was found between the scenarios. These results suggest that, apart for the edge-based version, the scenario did not influence the perceived and real latencies.

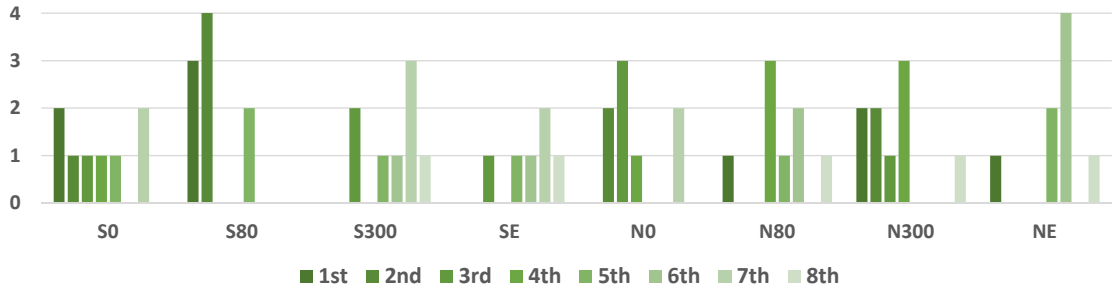


Figure 5.4: Distribution of conditions ranking.

Finally, to determine whether participants perceived the same latency as the real one, we computed a last series of Kruskall-Wallis statistical tests with multiple comparisons (n=8, α=0.05), which are reported in the rightmost column of Table 5.2. Five conditions out of eight, i.e., S80, S300, N0, N80, and N300, revealed a significant difference between the latency perceived by participants and the real latency measured by the system. In these cases, these results suggest that the participants did not perceive the real latency and were wrong in their perception. On the contrary, there is no statistically significant difference between the perceived latency and the real one for three conditions: S0 (probably because this condition represents the typical cloud-based configuration), NE, and SE (which are exactly our two challenging conditions for edge-based computing).

5.2.3 Games Ranking

In this section, we removed one outlier out of the ten participants as he expressed his preference not by as a ranking of conditions but as a preference for the non-shared scenario over the shared scenario. Fig. 5.4 depicts the distribution of all ranks expressed by participants for the eight conditions. Fig. 5.5 compares these distributions to identify for each place (ranging from the first place to the eighth place) which condition is expressed as the best ranked. The S80 condition is ranked the most frequently for the first, the second, and the fifth places, thus representing the condition that is the best placed overall. The N0 condition is ranked the most frequently for the third place, while N80 and N300 are *ex aequo* for the fourth place. The NE condition is ranked the most frequently for the sixth place and the S300 condition for the seventh place. For the last place, most conditions received the same low ranking, thus suggesting that they did not identify any preference for this place. Overall, participants reported that this ranking was very challenging to them as they did not identify any particular strength or weakness of a condition over another one, apart from the perceived latency that was discussed in the previous section.

Figure 5.5: Best ranked conditions per place (from 1st to 8th place).

We believe that the classification done by participants, which is very varying, cannot express a precise representative preference. Due to the small population sampling (n=9), statistical tests were not significant for the ranking of the games. But the notes and remarks expressed by the participants help to precise some aspects. One of the recurring notes was that they experienced difficulties in differentiating the external aspects of the games. This could suggest that the Pictionary application did not modify their user habits. They were not able to see the difference between edge and cloud versions.

5.2.4 Informal Comments and Observations

Scenarios and Devices. We tested two scenarios. To induce some device diversity, each participant brought her own device. We can distinguish two main categories: mobile platforms (such as smartphones and tablets) and stationary platforms (such as laptops and desktops). We found some correlation between the favorite scenario and the device category: 83.3% of the mobile participants prefer the shared scenario that leverages the low latency of the network. All participants prefer the non-shared scenario in general when no constraint is imposed on the network. The GUI is considered as a benefit for the device diversity and its characteristics.

Impact of Server Disconnection. We finished a game with an edge computing-based version and we shut down the server. At that time, participants reported that they even did not notice it and were still playing. In a discussion with them after the game, all participants confessed that they did not notice that the server was disconnected. This suggests confirming an advantage of edge computing over the cloud computing paradigm. The edge version works well in case of server disconnection and it is not noticed by end-users.

Fun with the Game. Multiple players seem to enjoy playing and sometimes winning Pictionary. Some of them were focused on victory and said that they enjoy playing.

5.3 Conclusion

In this chapter, we showed that the Pictionary interactive application developed for the purpose of testing edge computing is not less usable than average interactive applications (SUS OVERALL SCORE=69.72), with a fair agreement among

participants and a very good reliability among them. The two edge computing-based versions, with shared scenario and non-shared scenarios, were subjectively perceived as the versions benefiting from the best latencies among all conditions, including all cloud computing-based versions with or without delay. These two perceived latencies did not reveal any significant difference with the real latencies measured by the system throughout the experiment. This was also the case for the cloud computing-based version with the shared scenario without any delay. In this evaluation, we designed an experiment controlling a maximum of variables while keeping a realistic scenario to preserve ecological validity. The result analysis shows that new scenarios leveraging the low latency of edge computing to detach part of GUI are favorable to the device diversity and their varying characteristics. The assumptions made at the beginning of this thesis are validated.

Chapter 6

Conclusion

6.1 Summary of the Contributions

The goal of this thesis was to design a edge computing adaptation framework and implement a proof of concept to check our assumptions. We organize the work in multiple steps:

1. The related work, what are the existing alternatives and define the challenge
2. Design of adaptation of a cloud application into an edge computing based
3. Implementation of a proof of concept
4. Evaluation and assumptions testing

In chapter 2, we talk about different alternatives of cloud and edge computing. Cloud presents good performance for heavy calculation, but the high latency and bandwidth bottleneck are limitation which require new solutions. Computer scientists leverage low latency to create GUI detaching. We define edge computing as the migration of the host to the user device.

In chapter 3, we design the conceptual framework to adapt a generic cloud computing based application insight of developing a proof of concept. We focus on stateful interactive applications. The adaptations of the cloud application are of two types, modification and addition. The adaptation of the cloud communication mechanism is only developed for event-based information exchange. The behavior linked to the event sent or received by the host must be adapted to ensure the local behavior of the device hosting the application. The design of edge computing based application needs to add a migration protocols of the host. Depending the application requirement, a state consistency protocols can be also added. The

graphical user interface can be distributed to fit the diversity of devices. The separation in blocks to detach parts of GUI on multiple devices helps to propose multi models for GUI.

In chapter 4, we create a proof of concept using the conceptual framework of chapter 3. First, we described the criteria to found the cloud application. After, we described the application and his mechanisms. To complete the adaptation from cloud computing to edge computing, we modify the mechanisms and add the edge computing needed mechanisms. Finally, we separate the GUI in three blocks: one to drawing, one to display the drawing, and the last with the role of the users.

In chapter 5, we test the applications in a concrete experimentation. The users have difficulty to distinguish differences between cloud and edge computing applications. The second scenario based on GUI splitting was preferred by almost all smartphone users. The users do not notice that the server stops running during the game with the edge version. The latency in of edge application is lower than the cloud application latency. Edge computing is a real efficient alternative to cloud computing.

6.2 Benefits and Shortcomings

Our proof of concept demonstrate multiple benefits of moving the server parts on the user device. In our definition of edge computing, the edge computing takes advantage of the large computing power available at the edge of the network. The service can still work in case of server disconnection. The interface proposed by cloud applications can runs in edge computing without impacting the user experience. From the usability point of view, edge is comparable to cloud computing and can be considered as a real alternative for cloud computing. Many kinds of applications can be adapted to work with edge computing. Thanks to low latency, new GUI can be created by separating GUI in blocks and distributing them on multiple devices.

Edge computing have also shortcomings. Our definition of edge computing involves devices connected in a peer-to-peer way helped by a small server. The libraries providing these ability are not plenty as explained in the paragraph 3.2.4. And these libraries are not optimal to adapt a cloud computing based application into an edge computing based. They complexify too much implementation or present multiple bugs limiting the possibilities of edge computing (e.g.: Socket.io-p2p

can not support a high number of connected user in p2p). The adaptation of an application may need architecture choice depending the requirement of the application. e.g.: host task separation or state consistency mechanism. The last shortcoming is the code is more complex to implement because it must manage more tasks.

6.3 Future Work

Other potential future works can be done in the short term. Other kinds of hosting separation can be tested. For example, creating multiple rooms in a pictionary application and test the efficiency of a whole-for-multi host separation. Test a state consistency mechanism and its reliability.

More complex tasks can also improve the edge computing solution. Create an automated management of the host migration depending of some criteria can be a real improvement to optimise the usage of the computing power of user devices. Implementing or improving the libraries providing the ability of managing the peer-to-peer connection assisted with a server. These libraries must be simple to use to avoid useless complexification of the implementation.

List of Figures

List of Tables

Glossary

GUI Graphical User Interface.

IoT Internet of Things.

MEC Mobile Edge Computing.

QoS Quality of Service.

UI User Interface.

Bibliography

[1] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[2] C. Li, Y. Xue, J. Wang, W. Zhang, and T. Li, "Edge-oriented computing paradigms: A survey on architecture design and system management," *ACM Comput. Surv.*, vol. 51, no. 2, Apr. 2018. [Online]. Available: https://doi.org/10.1145/3154815

[3] Y. Ai, M. Peng, and K. Zhang, "Edge computing technologies for internet of things: a primer," *Digital Communications and Networks*, vol. 4, no. 2, pp. 77 – 86, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2352864817301335

[4] S. Wang, J. Xu, N. Zhang, and L. Yujiong, "A survey on service migration in mobile edge computing," *IEEE Access*, vol. PP, pp. 1–1, 04 2018.

[5] J. Ren, D. Zhang, S. He, Y. Zhang, and T. Li, "A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet," *ACM Comput. Surv.*, vol. 52, no. 6, Oct. 2019. [Online]. Available: https://doi.org/10.1145/3362031

[6] D. Grolaux, J. Vanderdonckt, and P. Van Roy, "Attach me, detach me, assemble me like you work," vol. 3585, 09 2005, pp. 198–212. [Online]. Available: https://www.researchgate.net/publication/221054087_Attach_Me_Detach_Me_Assemble_Me_Like_You_Work

[7] J. Melchior, D. Grolaux, J. Vanderdonckt, and P. Van Roy, "A toolkit for peer-to-peer distributed user interfaces: Concepts, implementation, and applications," *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems EICS'2009*, 2009. [Online]. Available: https://www.researchgate.net/publication/220728556_A_model-based_approach_for_distributed_user_interfaces

[8] Response times: The 3 important limits. [Online]. Available: https://www.nngroup.com/articles/response-times-3-important-limitsaddendum=

[9] K. Gyarmathy. Edge computing vs. cloud computing: What you need to know. [Online]. Available: https://www.vxchnge.com/blog/edge-computing-vs-cloud-computing

[10] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the internet of things realize its potential," *Computer*, vol. 49, no. 8, pp. 112–116, 2016.

[11] E. Rivière, *Cloud computing LINGI2145 UCLouvain*, 2020.

[12] Cloud, fog and edge computing – what's the difference? [Online]. Available: https://www.winsystems.com/cloud-fog-and-edge-computing-whats-the-difference/#:~:text=Fog%20computing%20uses%20a%20centralized,interfacing%20to%20sensors%20and%20controllers.

[13] Moore's law. [Online]. Available: https://www.britannica.com/technology/Moores-law

[14] A. Kouloumpris, T. Theocharides, and M. Michael, "Metis: Optimal task allocation framework for the edge/hub/cloud paradigm," pp. 128–133, 05 2019. [Online]. Available: https://www.researchgate.net/publication/332282239_Metis_Optimal_Task_Allocation_Framework_for_the_EdgeHubCloud_Paradigm/citation/download

[15] Fog computing vs. cloud computing for iot projects. [Online]. Available: https://www.sam-solutions.com/blog/fog-computing-vs-cloud-computing-for-iot-projects/

[16] Multi-access edge computing (mec) and distributed cloud. [Online]. Available: https://www.sdxcentral.com/edge/definitions/whats-the-difference-between-mec-and-distributed-cloud/

[17] R. buissness. What are the benefits of edge computing vs. cloud computing? [Online]. Available: https://www.rcn.com/business/insights-and-news/insights-articles/edge-computing-vs-cloud-computing/#:~:text=The%20main%20benefits%20of%20edge,Reliable%2C%20uninterrupted%20connection

[18] Edge computing vs. fog computing: What's the difference? [Online]. Available: https://www.cmswire.com/information-management/edge-computing-vs-fog-computing-whats-the-difference/

[19] Cloud, fog, and edge computing: 3 differences that matter. [Online]. Available: https://dzone.com/articles/cloud-vs-fog-vs-edge-computing-3-differences-that

[20] S. Taherizadeh and V. Stankovski, "Auto-scaling applications in edge computing: Taxonomy and challenges," in *Proceedings of the International Conference on Big Data and Internet of Thing*, ser. BDIOT2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 158–163. [Online]. Available: https://doi.org/10.1145/3175684.3175709

[21] A. Reiter, B. Prünster, and T. Zefferer, "Hybrid mobile edge computing: Unleashing the full potential of edge computing in mobile device use cases," in *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, ser. CCGrid '17. IEEE Press, 2017, p. 935–944. [Online]. Available: https://doi.org/10.1109/CCGRID.2017.125

[22] G. Tato, M. Bertier, E. Rivière, and C. Tedeschi, "Sharelatex on the edge: Evaluation of the hybrid core/edge deployment of a microservices-based application," in *Proceedings of the 3rd Workshop on Middleware for Edge Clouds amp; Cloudlets*, ser. MECC'18. New York, NY, USA: Association for Computing Machinery, 2018, p. 8–15. [Online]. Available: https://doi.org/10.1145/3286685.3286687

[23] Facebook edge application. [Online]. Available: https://research.fb.com/publications/taiji-managing-global-user-traffic-for-large-scale-internet-services-at-the-edge/

[24] galdinorosas, "https://github.com/galdinorosas/pictionary," 2016.

[25] Pictionnary rules. [Online]. Available: https://www.regles-de-jeux.com/regle-du-pictionnary/

[26] Pictionnary pict. [Online]. Available: https://www.cdiscount.com/juniors/jeux-de-societe-cartes/pictionary-classique-mattel/f-12079-ma55847.html

[27] J. Brooke, "SUS-A quick and dirty usability scale," *Usability evaluation in industry*, pp. 189–194, 1996.

[28] A. Bangor, P. T. Kortum, and J. T. Miller, "An empirical evaluation of the system usability scale," *International Journal of Human–Computer Interaction*, vol. 24, no. 6, pp. 574–594, 2008. [Online]. Available: https://doi.org/10.1080/10447310802205776

[29] A. Bangor, P. Kortum, and J. Miller, "Determining what individual sus scores mean: Adding an adjective rating scale," *J. Usability Studies*, vol. 4, no. 3, p. 114–123, May 2009.

[30] J. Cho, K. Sundaresan, R. Mahindra, J. Van der Merwe, and S. Rangarajan, "Acacia: Context-aware edge computing for continuous interactive applications over mobile networks," in *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 375–389. [Online]. Available: https://doi.org/10.1145/2999572.2999604

[31] J. Melchior, J. Vanderdonckt, and P. Van Roy, "A model-based approach for distributed user interfaces," *EICS : Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*, 06 2011.

[32] ——, "Distribution primitives for distributed user interfaces," 11 2011.

[33] U. Braga Sangiorgi, V. G. Motti, F. Beuvens, and J. Vanderdonckt, "Assessing lag perception in electronic sketching," in *Proceedings of the 7th Nordic Conference on Human-Computer Interaction: Making Sense Through Design*, ser. NordiCHI '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 153–161. [Online]. Available: https://doi.org/10.1145/2399016.2399040

[34] Y.-S. Seo and J.-H. Huh, "Gui-based software modularization through module clustering in edge computing based iot environments," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–15, 09 2019.

[35] P. Millecamps, "Les data centers, un gouffre énergétique ?" 2020.

[36] Internet : le plus gros pollueur de la planète ? [Online]. Available: https://www.fournisseur-energie.com/internet-plus-gros-pollueur-de-planete/#:~:text=L'envoi%20d'un%20mail,Garder%20seulement%20les%20emails%20n%C3%A9cessaires.

[37] Comment utiliser le system usability scale (sus) pour mesurer l'expérience utilisateur ?

[38] T. Taleb, A. Ksentini, and P. A. Frangoudis, "Follow-me cloud: When cloud services follow mobile users," *IEEE Transactions on Cloud Computing*, vol. 7, no. 2, pp. 369–382, 2019.

[39] Z. Zhou, X. Li, and G. Sun, "Accelerate service live migration in resource-limited edge computing systems," in *Proceedings of the 4th*

ACM/IEEE Symposium on Edge Computing, ser. SEC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 354–355. [Online]. Available: https://doi.org/10.1145/3318216.3363456

[40] I. Hadžić, Y. Abe, and H. C. Woithe, "Edge computing in the epc: A reality check," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, ser. SEC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3132211.3134449

[41] M. Zorrilla, N. Borch, F. Daoust, A. Erk, J. Flórez, and A. Lafuente, "A web-based distributed architecture for multi-device adaptation in media applications," *Personal and Ubiquitous Computing*, vol. 19, pp. 803–820, 2015.

[42] A. Domínguez, J. Flórez, A. Lafuente, S. Masneri, I. Tamayo, and M. Zorrilla, "A model for user interface adaptation of multi-device media services," *IEEE Transactions on Broadcasting*, pp. 1–13, 2021.

Appendix A

Code of proof of concept

Code of the different versions of the application can be found in the repository: https://github.com/dewanetout/Pictionary.git To reproduce the different scenarios, the settings of the drawing device and display device must be set by the user. The instruction to launch the different versions are explain in the README file.

In the cloud version files, you can found a traffic controler (`tc.js`).

Appendix B

Summary of the event in cloud application

Mechanism	Event	Variables	Receiver	Description
Connection	`connection`	socket	Server	Event sent when a new client is connected to the game
	`setID`	ID	Client	Event sent to the client to advertise his unique ID
	`wait`		Client	Event sent to the client when he is the only connected
	`play`		Clients	Event sent to all the clients when the new client connected is not the first connected
	`playerJoined`	Number of player	Clients	Event sent to all the clients to informed that a new player is connected
Drawing	`serverToClient`	Point	Clients	Event sent to inform clients of a new point to draw
	`clientToServer`	point	server	Event sent to inform the server that the client draw a new point
Guessing	`GuessToServer`	word	server	Event sent to the server to inform the trial of guessing
	`GuessToClient`	word(s)	clients	Event to the client to inform the trial to guess and if its right the word that has to be guess
User disconnection	`Disconnect`		server	Event sent to inform the disconnection of the player linked to the socket
	`drawerHere`	boolean	server	Event sent to the server to inform the role of each client (true if drawer)
Other	`Winner`	ID	Clients	Event to inform all the clients of the victory of the player identified by ID
	`userTypeCheck`	Boolean	Clients	Event sent to all the client to set the user role (true if drawer, otherwise false)
	`clientToServe -rWordCheck`	Word	Server	Event sent to inform the server of the word to guess

Table B.1: Summary of event send in the cloud version

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**