Edge node federation for a local cloud : asynchronous protocols, data replication and energy consumption - A state of the art

Ralph Hatoum

2023

1 Introduction and context

Cloud computing has drastically changed the IT landscape by allowing people and companies to offload their data storage and computing needs to data centers ran by specialized companies. Running a data center is an intensive task, as administrators have to ensure resources are safe, reliable and available when needed by customers. At the same time, the operation needs to be energy efficient, especially given the current price and scarcity of electricity.

While cloud computing is now a widely used and deployed solution that has proven its worth, the communication between users at the edge of the network and a distant data center can be time consuming. This makes cloud computing unsuitable for delay sensitive applications.

Edge computing aims to solve this problem by adding computing and storage capabilities closer to users, at the edge of network. These nodes are constrained (less powerful, less storage capability, restricted access to energy compared to a data center), but their proximity to users would reduce the delay incurred by communication. Fog computing aims at subordinating this edge layer to a distant data center, allowing direct communication between these layers.

The Internet of Things (IoT) is another of the motivation behind adding computing capabilities at the edge of the network. The billions of connected devices will cumulatively generate terabytes of data everyday, that would flood networks if it was to travel up to a data center to be stored. Having the capability to store data and process it at the network edge would therefore be beneficial in this case.

On the other hand, another issue raised by data centers is data ownership and sovereignty. Most data centers are run by a handful of companies, in a handful of countries. Certain companies might want to retain ownership of their data, or at least have the guarantee that their data will stay within their own country or region, while receiving similar service level in a data center (automatic replication of data, resilience to failure, high availability).

Edge and fog computing therefore open the path to new decentralized data center architectures. Companies could own a node, to store data and perform their computing needs. This node could be interconnected with other nodes on a regional level, allowing for data replication for backup purposes and task offloading in case some nodes are overloaded. It would be more resilient to failures, as all nodes are geographically distributed within a given distance radius, so there is no risk of a site physical incident (fire or flooding). This architecture allows for regional level data and computing power sovereignty, as customers would mostly be served by their proper node, and have their data replicated only within a given region. It also allows for reduced delay, as users would only communicate with nodes within their own region.

However, this architecture also comes with its share of challenges. First off, nodes are interconnected through the internet, making communication between them unreliable. They are also constrained in available storage space, computing power, and energy. This architecture management should also ideally be done in a decentralized, peer to peer manner, as a centralized network controller would constitute a single point of failure.

We'll examine the current state of the art of edge-fog computing and peer to peer systems in order to provide data center like service on the described architecture. For that, we'll analyze main challenges and their current solutions in the management, load balancing and data replication in a data center. Then, we'll look at how peer to peer systems allow for high data availability and communication between nodes. Finally, we'll look at the current solutions to these challenges in an edge-fog layer, in order to examine relevant research directions.

2 Data center consolidation and data replication

In a data center, there are tens of thousands of machines available for data storage and computing tasks. The company operating the facility has service level agreements (SLA) with customers, which are contractual quantitative agreements about data availability, resistance to failure, response time and sometimes more. Resources available have to be split among customers, taking into account their needs and SLA. At the same time, servers cannot constantly stay on as it would drive energy costs up. The system therefore needs to be able to adapt to the workload it is given, and dynamically boot or stop machines, in order to keep energy costs down while making sure customers are experiencing the level of service they have paid for.

2.1 Virtualization

In a data center, storage and tasks are not performed directly on physical machines, but in virtualized environments. This is convenient for several reasons. First, it can allow for isolation when several customers are receiving service from a single physical machine. Second, it makes migrating tasks from a physical machine to another much easier. The virtual environment can be stopped, transferred to the other machine, and restarted there. Two main kinds of virtualization are to be considered : virtual machine (VM) based and container based.

In the case of VMs, the machine is fully virtualized, meaning there exists a virtual version of each physical computer component (CPU or memory for example). These VMs run on top of a physical machine, running itself a hypervisor responsible for allocating physical ressources to VMs. The VM runs an OS oblivious of virtualization. Container based virtualization happens at the application level. The physical machine is running an OS kernel, in which a container engine runs. Containers run inside that container engine. This means they can share files and dependencies if needed.
Both technologies are widely used, and extensive research exists on their performance in different scenarios.

In [19], Potdar et al. provide us with extensive performance comparisons between VMs and containers running on Docker, an open source container engine. They tested CPU performance, memory throughput, disk I/O time, maximum load and operation execution time. In every single one of these tests, the Docker container outperformed the virtual machine. They explain that the hypervisor seems to cause significant delay when performing operations on a virtual machine. In order to rigorously determine if there is a critical performance difference between virtual machines and containers, they performed a student's t-test, that confirmed the initial observations. However, it is important to insist on the fact that this gain in performance comes at the cost of potential security breaches, because containers files. The authors in [19] consider this fact in their future work section, where they plan on working on more secure containers.

These two technologies should not be thought of as opposites, as several papers show combining both can solve the security issues while maximizing performance. In [11], authors benchmark performances of a system running containers in a VM, in order to quantify the overhead incurred by the two layers of virtualization. They performed tests on different VMs, running different OSes (CoreOS, RancherOS, CargOS, Atomic Host and Alpine OS). These are all minimalistic OSes, and with the exception of Alpine OS, they are container-centric, meaning they are designed to efficiently run Docker and Docker containers. For the comparison, they also ran all the same tests on bare-metal. These tests measured CPU, memory, disk read and write performance and TCP transmission rates. Combining all these observations, they came up with optimal virtual machine and host OS combinations for CPU intensive, Memory intensive, Disk I/O intensive, and network intensive applications. This approach therefore helps containers with their security and isolation issues, but that comes at a performance cost, that can be acceptable if the right VM-guest OS combination is chosen.

2.2 Load balancing

In order to balance the workload over the different physical machines available, virtualized environments are migrated from a physical machine to another. In this section, we'll take a closer look at how these migrations happen, but also how the system decides whether a migration should happen or not, and between which two machines.

Migrating virtualized environments Migrations have a cost that mostly takes into account delay and energy consumption in a data center, as virtualized environments have to be stopped, transferred over the data center's network, and restarted on the new machine. While we can assume this network to be fast and reliable, in moments of high usage the available bandwidth can shrink and affect migration time. Several papers have assessed the cost of migrating VMs and containers.

In [3], authors describe the "pre-copy" approach to migrating VMs from a source physical machine to a destination physical machine. It consists of a first phase where the memory content of the VM is iteratively copied from the source to the destination machine. The VM is then stopped, migrated, and resumes its work, with its memory content, on the new host. Authors tested their approach on a two server testbed. They showed that as the available bandwidth on the link increased, the power consumption of both servers increases as well. They explained this result by the fact that the faster the communication is, the busier the hypervisors get. They conclude that : 1) the migration time increases linearly as the VM's RAM size increases and 2) the migration time decreases exponentially as the available bandwidth increases. However, we will insist on the fact that this decrease in migration time happens at the cost of an increased energy consumption.

On the other hand, [6] presents Voyager, an approach to container migration for stateful services. The authors note most of an application's downtime is due to data copy. They therefore propose the idea of migrating the container to the target machine first, and without the data. Then, through efficient data federation, the container on the target machine can remotely read its data on the source machine when it needs it, and lazily write it on the target host. To achieve this, while the application is running at the source host, a union mount is created between the source and target hosts' file systems. This approach was tested on two machines in a data center. The authors estimate the downtime of an application to be 2-3 seconds at most with Voyager, with room for improvement. As for performance overhead, the most overhead is observed during the federation state, where the migrated container has to access files remotely, where we can observe up to 75% overhead. However, once these files are read, they are stored on the target host's local memory, which means this overhead won't be experienced on future read operations on this data chunk. Therefore, Voyager allows for more efficient container migrations by not copying all data first, but rather lazily copying it from the source host when it is needed.

Algorithms for load balancing We have established how virtualized environments are migrated from a machine to another, but we haven't looked at why and when these migrations should happen. Ideally, data centers need an algorithm to find the optimal mapping of virtualized environments on the available physical machines in the data center, while optimizing parameters like energy consumption, delay or task blocking probability. Finding such a configuration is a NP-hard problem; so heuristics and meta heuristics are used to find near optimal solutions to this problem.

In [8], Marotta et al. aim at optimizing the power consumption of a data center by taking into account the servers' power consumption, the network equipment's power consumption and the number of migrations. Given a set of active servers running VMs, a set of active switches and their respective power consumption, they aim at finding a set of migrations that will define new sets of active servers and active switches, that will satisfy the required computing needs of customers, while minimizing the overall energy consumption and the overhead caused by migrations. They define an objective function, which is a weighed sum of three terms : one for servers' power consumption, one for the number of migrations and one for the switches' power consumption. The problem is modeled as a mixed integer linear programming problem, in order to minimise the previously presented objective function. This problem is NP-hard, so they present a meta heuristic called Simulated Annealing based Resource Consolidation (SARC), which according to the authors is a simplified version of simulated annealing. The aim is to start with an initial configuration (the current configuration), then iteratively explore new configurations by changing the initial one and evaluating their effect on the objective function. By adjusting the weights in the objective function, we can obtain solutions that prioritize certain parameters over others. Therefore, this paper gives us one approach at a load balancing algorithm to optimize energy consumption and the number of migrations.

2.3 Load prediction

As previously stated, data center operators' main concern is respecting SLAs, while consuming a reasonable amount of energy. Previously presented load balancing approaches help find new suitable configurations, but these approaches could benefit from future load forecasts to anticipate peaks and adapt faster. To this extent, we'll take a look at a load prediction method to anticipate peak demands and consolidate machines faster.

In [14], authors tackle load prediction with deep learning. The main challenge they identify is the highly variable nature of the number of incoming requests, as well as the high dimensionality of data available. To solve the former, they propose a method based on the use of a recurrent neural network (RNN), that they call L-PAW. This allows for historical memories to have weight in the predictions. To solve the latter, authors aim at extracting important features to reduce the data's dimension, while outputting a vector as close as possible to the input, thanks to a Top Sparse Autoencoder (TSA). The overall prediction pipeline goes as following : data is collected from real data center activity and goes through a pre-processing step for normalization. Then, it goes through the TSA model. The data with reduced dimension is then fed into the L-PAW model, that can output a prediction of future load, but also learn historical tendencies. Real data center data is then collected again and goes through the same process. The approach was tested on three types of workload, but the most import one is the highly random workload. Predictions over different time lengths (seconds to days) were computed, and L-PAW was successful in predicting peaks of requests.

2.4 Data replication strategies

As previously detailed, one of the services that can be offered by a data center is data storage. Here, customers pay to have their data stored on machines in the data center, with the guarantee that their data will be highly available, and backed up in case of a failure. Considering hard drives can fail, it is important to have data replicated on different physical devices, in order to make sure chunks of data are never lost. Having multiple copies of the same data can also help us achieve increased availability. Moreover, it seems reasonable to think that data should not only be stored in multiple servers in one data center, but also across multiple data centers. However, saving several copies of the same data consumes more resources. It is clear that a data center needs an efficient data replication strategy, to ensure data security and availability while ensuring a reasonable energy consumption. In [23], an approach to dynamic data replication is presented. They consider a hierarchical architecture, where smaller data centers are interconnected to bigger, more reliable ones. Data files can exist in several copies, not only over all data centers, but also within a single data center. Data centers are classified in layers, from most reliable (biggest) to least reliable, and each layer is attributed a cost. This cost takes into account metrics like reliability, performance and speed. Authors model the problem as a knapsack problem, where data replicas need to be placed across and within data centers to minimize the overall replication cost. This problem is NP-hard, and they present their heuristic base on two algorithms. The first one is a multi objective particle swarm optimization (MO-PSO) based approach, and it is used to determine which files should be replicated, and when. The second algorithm is based on the multi objective ant colony optimization (MO-ACO) method. This algorithm is used to find an optimal replica placement scheme. The MO-PSO and was compared to two other approaches : Elastic File System (EFS) replication and DCR2S. Results show that as the number of cloudlets increases, the MO-PSO approach performs better than the other 2 approaches. MO-ACO has shown to reduce paths lengths to retrieve data, and reduced data loss rate to near 0 has the number of nodes increases.

3 Peer to peer systems

Peer to peer systems are network architectures where all nodes are interconnected, and can exchange information and files when needed; there is no client and server role. In this section, we'll first look at how peer to peer systems and edge computing are similar fields. Then, we'll take a look at how peer to peer systems allow for data replication and efficient retrievals. We'll also look at other useful advances in the field of peer to peer systems, especially energy trading.

3.1 Similarities with edge computing

Several papers examine the similarities between peer to peer systems and edge computing. While they aim to achieve different goals overall, it does seem like a lot problems that edge computing brings on the table also are issues found in a peer to peer system, so taking advantage of the decades [10] of research on these systems is important.

In [13], authors examine these similarities. Peer to peer nodes are usually constrained in resources, and can frequently go on and offline. Efficient data storage and retrieval are also problems tackled in peer to peer research. Nodes are also owned and managed by different agents in both cases. Peer to peer nodes are subject to security threats, such as DDoS attacks, just like edge computing nodes. It therefore makes sense to study peer to peer systems, as they have provided answers to all these problems. While these solutions might not directly apply to an edge and fog layer, taking inspiration from them could be beneficial. In [10], authors present different possible use cases of peer to peer interactions between edge computing servers, as well as possible research directions. For example, authors highlight a scenario where multiple users using the same services could have similar computation requirements. In this case, peer to peer interactions are a solution to share the computation results, resulting in less energy consumption and better response time. Another scenario, in certain ways similar to the architecture we are considering, is one where users are served in priority by their closest edge server, that forwards the request if needed to its other closest servers, ensuring the user is served by a relatively physically close server. Authors also point out routing and security as emerging fields that have not been successfully solved in peer to peer nor edge computing literature. On another hand, they claim edge computing could take advantage of the proximity awareness systems proposed in peer to peer architectures to get in contact with their neighbors. All this is to say that peer to peer systems and edge computing have similarities and several papers explore how both fields can be bridged.

3.2 Peer to peer file system and storage

One of the most well known peer to peer file system is the IPFS, Inter planetary file system. In [2], its creator details how it works. IPFS brings together a set of already proven, working technologies to build an efficient distributed file system. Nodes have a public and a private key, and each have an ID generated by hashing the public key. These keys allow for nodes to check for messages' authenticity as they communicate with other ndoes in the network. For routing, IPFS uses a distributed hash table to allow efficient retrieval of data based on keys, and efficient distribution of data. Similarly to BitTorrent, IPFS stores data by splitting it into blocks and distributing it on the network. Nodes are incentivized to store data blocks needed by their peers and free riders are blacklisted thanks to a debt ratio that acts as a reputation metric. If a node has successfully shared and download data before, it will be known as a trusted node. This is also used a way to prevent - or at least discourage - nodes to disconnect from the network, as this would mean resetting their trust metric and losing network privileges. IPFS offers a lot of possibilities regarding file versioning and content addressing. While we won't get into all features here, what we should keep in mind regarding IPFS is that it is highly customizable and designed for applications to be built on top of.

In [20], authors give their approach at decentralized storage using blockchain. We suppose a network of peers who make their storage capabilities available to users. When a user wants to store files on the system, he will need an Ethereum wallet. The file he wants to store is first encrypted with his wallet key, and sent onto a smart contract. This contract will require the user to pay a certain amount of funds. Once this is validated, the file will be split into blocks, and distributed among peers in the network, to make several copies. These files are encrypted, and can only be retrieved with the user's wallet key, and funds paid by the user are split among peers who store the file. The file can be accessed by the user whenever needed, and peers who fail to provide the file will be blacklisted. This approach makes it so that peers are compensated for storing files that aren't theirs. Using a cryptocurrency to reward users that store data and are reliable is relevant to our study. However, the time required to upload a file is relatively high, and so will be the time to retrieve it.

In [26], authors present an approach at dynamic data replication in a peer to peer network. Each data block's popularity is calculated, as popular data blocks should be replicated to make sure they are easily accessible by nodes in the network, and less popular data blocks should exist in less copies to free storage space on the system for new data. The popularity metric is computed based off the number of requests to access the data block. A minimum number of data replicas is calculated, taking into account an availability threshold, the probability of failure of nodes and offline hours of nodes. Whenever a data block's popularity falls behind, the number of replicas will go back down to the minimum number of replicas. Then, authors present an approach to determine the right nodes to store data on. For this, they give each node a score, based off its features, efficiency in the network, and position in the network.

3.3 Load balancing in peer to peer energy trading networks

While load balancing in energy trading and edge computing are not exactly the same problem, analyzing the former is beneficial to our study as there is extensive literature available. Because energy is not easy to store efficiently, electricity providers and consumers who have the means to produce energy need to adapt their production to the current demand at any given moment. Given this demand is not easy to predict, and that if electricity is not used or stored, it will be wasted, the smart grid was introduced. It allows energy producers to sell leftover production to anyone else on the grid. For a while, this was done in a centralized manner [15], but research is increasingly leaning towards decentralized approaches. We could make an analogy with cloud and edge computing, where a provider who has unused resources could be willing to make them available.

In [15], authors present a decentralized peer to peer energy trading market based on blockchain. Here, smart buildings collect and can classify themselves into 3 categories : energy surplus, energy deficit or equilibrium. This data is sent to a virtual layer, where a blockchain is implemented. Network agents can here decide if they want to sell energy or need to buy it. A smart contract takes care of verifying if the energy transfer is valid, and authorizes it. Authors have therefore created a trustless energy market where available power can be sold, but also where the price for the last transaction is saved, and used to determine the current market price of energy. Just like energy, computing power and storage that is not used by a peer could be made available to other peers for a price. Again, this price could take the form of a certain credit metric we would have to define, that could be a reward for making computing power and storage available.

3.4 Churn management in peer to peer systems

Peer to peer systems rely on having a fair number of nodes active on the network. While nodes can join the network as they wish, they can also leave it (whether it is intentionally or because of a failure) and join it again. This phenomenon is referred to as churn, and peer to peer systems have to efficiently deal with it.

In [1], one example of an approach against churn is presented. Authors notice previous literature mostly covers faults in a static manner : a random but bounded number of nodes are crashed, then the system is given time to converge back to a stable state. They consider this is not realistic, and they design a peer to peer network and an adversary agent, whose role is to randomly crash and add nodes in the network. The network is basically constantly failing, and is constantly converging towards a stable state, making it resilient to failures. For that, they distribute peers evenly on a d-dimensional hypercube. Each vertex of this hypercube represent a group of peers in the network. Each data block that needs to be stored on the network is assigned to one of the hypercube vertices, and it is ensured at least one of the peers in the vertex holds the data. The hypercube's dimensions varies as the number of nodes change in the network. Periodically, peers evaluate the number of nodes in the network by sending messages to each other (containing their ID and the ID of its adjacent peers) and the hypercube's layout and dimension are updated. However, it is important to highlight that this system is only effective for fail-stop failures, so under the hypothesis that there are no byzantines. Also, authors consider the time needed for peers to communicate can be bounded.

4 Edge and fog computing

Edge and fog computing bring computing power and storage closer to users. Nodes are geographically distributed, connected over the internet, restricted in computing power, energy and storage space, and most likely less reliable. Even under these constraints, we still want to achieve efficient use of resources, balance the load over nodes and achieve efficient and resilient file replication. We'll therefore explore these points with our new important constraints, and navigate through the proposed approaches to solve said issues.

4.1 Virtualization in an edge-fog architecture

Virtualization in the edge and fog layers happens for similar reasons as in data centers : to offer isolation and to make migrations easier. Most important points regarding virtualization were already exposed in the data center consolidation section. Here, we'll only cover the more specific point of virtualization on constrained nodes.

In [9], Morabito et al. aim at analyzing the use of different virtualization technologies for the Internet Of Things (IoT). In this case, nodes are usually extremely limited in computing power and storage. Virtualization is important in an edge computing environment, for the reasons we've exposed before, but also because nodes can be heterogeneous, so being able to make abstraction of the hardware layer facilitates service deployment. Particularly, they focus on the use of containers and unikernels. Containers have already been covered in the virtualization for data centers section; we'll simply summarize by saying they allow for virtualization at the application level, cause less memory and performance overhead than VMs and are faster to start, but are also harder to isolate as containers share files. Unikernels, on the other hand, are single-purpose kernels, that also allow for virtualization at the application level, and are constructed using virtual library operating systems. They consist in only keeping the parts of the OS necessary to run the application. This makes them lightweight and secure, and are definitely an appropriate choice to run on edge computing nodes. They are faster to boot and more lightweight than containers, and are a great choice for stateless applications. The paper goes on to examine the use of containers and unikernels in different possibles applications of edge computing (for example vehiculars networks or smart cities), and explains open issues. We won't get to all these points as we would fall out of the scope of this paper, but there are a few observations that are useful to our study. The authors hint at the fact that again, just like we covered with VMs and containers, a combination of containers and unikernels could be relevant in certain cases, especially to solve isolation problems and make orchestrations easier. Lightweight virtualization is also essential, as edge nodes might be shared by multiple providers. For inter-edge node communication, the authors deplore the lack of research regarding stateful application migration. Finally, regarding data storage, one of our main interests, authors hint at the fact that edge nodes are usually not a reliable place to store persistent data, and containers and unikernels are also not necessarily designed for this application. In such an architecture, data is usually stored in a centralized manner and loaded when needed. Persistent data in an edge architecture therefore deserves to be looked into and is a relevant scope of research.

4.2 Load balancing and offloading in edge computing

Load balancing is used to make the most out of available resources. In edge computing, this will mostly be about relieving overloaded nodes, by migrating virtual environments from a node to another. The difference with cloud computing is that those nodes are connected over the internet, meaning migrations will be more time consuming.

In [18], authors propose using intermediary nodes to distribute tasks. Intermediary nodes store edge nodes' physical attributes, such as CPU and memory capacity, but also monitor their dynamic attributes (their current resource usage). With all this information, the intermediary load is able to compute a node's current load. Using this information, nodes are classified into three states with a Naive Bayes algorithm : light-load, normal-load and heavy-load. Now, whenever a set of tasks arrives at an edge node, this node emits a request to the intermediary, that will elect nodes classified in the lowest load class possible. The sets of tasks will now be shared among the node that received the task and the lightly loaded nodes. In order to choose how the tasks will be split among them, a completion time function is defined, that will need to be minimized. Different configurations are explored through a particle swarm optimization based algorithm; at the end of which the set of tasks is split into subsets that will each be performed on one edge node. They therefore achieve load balancing and task distribution. This approach uses a centralized controller, and allowing resources at edge nodes to be used for task computing. However, it also brings in a single point of failure, as the intermediary node can fail. This approach also systematically calls this intermediary node, and does not take into account the fact that, if a node is not overloaded, it might not need to offload its tasks.

Other approaches aim at utilizing game theory to achieve efficient load balancing. For example, in [25], authors propose a load balancing strategy between cloudlets in a mobile edge computing context, modeled as a competition, where users choose an offloading startegy to optimize their own needs. This is modeled as a stochastic congestion game with incomplete information; meaning users randomly offload their tasks to a cloudlet and establish which cloudlet is the best for them, without knowing anything about other users (not even knowing their existence). Authors prove this game has a Nash equilibrium, and propose a decentralized learning algorithm to find it. This work differs from the previous paper as it presents a decentralized approach, getting rid of the single point of failure problem. However, this work does not take into account users' locations and wireless transmission delays, and these two parameters could very much influence the decision to which cloudlet to offload to. But, these parameters do not play as big of a role outside of the mobile edge computing context.

In [21], authors try a load balancing approach with reinforcement learning. They present a healthcare monitoring network with an IoT layer, a fog computing layer and a cloud layer. We'll focus here on the fog layer; it is made up of fog servers that are split into different regions. Each region has a Master Server (MS), who has access to all fog servers characteristics, and is in charge of administrating the region, meaning all computation relative to load balancing and resource allocation is performed on the MS. First, an adaptive weight metric is calculated for each fog server, based on their capacity (RAM, CPU usage). These weights allow for fog servers to be classified in 3 sets : balanced, under loaded and overloaded. Whenever a new task arrives, it will be assigned to an under loaded node, and overloaded nodes will migrate some tasks to under loaded nodes : this is the job of the allocation agent (resource allocator, RA), and this where reinforcement learning is utilized. The resource allocator learns the select the best fog server to execute new tasks on the one hand, and learns which process is best to migrate on an overloaded node. The method has proven to reduce energy consumption and the number of migrations compared to other load balancing approaches. This is a take on load balancing in a centralized manner again, but this time it utilizes a machine learning technique.

In [5], a load balancing approach is presented on a software defined embedded network. The main difference brought in by this is the fact that there are memory servers shared by fog nodes. The paper also tackles task scheduling, more specifically where the tasks shall be performed (at edge node or at client), considering offloading to an edge node can end up being inefficient if said node is overloaded. Migration here is referred to as service placement; either a service is present on a machine or it isn't. If it is, it can be started and the response time will be smaller.

If a service isn't present on the node, it will need to be transmitted to this node before the computation can take place. This is where memory servers come in : they act as the system's file and memory system. Authors jointly optimize the problem of task offloading and service placement to minimize computation time. This problem is modelled as a mixed integer linear programming problem; but as solving this would be too costly, they propose their own heuristic. While the approach was only tested on a simulator, it does show good performance, and proves taking advantage between computing needs at the client level and fog level can prove to be efficient. The management of storage servers is relevant to our study, as this has not been presented in other papers.

In [17], authors present an attempt at an incentive driven edge computing offloading scheme. The context is a mobile edge computing architecture where users with edge devices (smartphones, computers, tablets) interact with the closest base station (BS). Edge devices and base stations have computing and storage capabilities, and the approach explores the idea of offloading tasks not only from edge devices to the base station, but also between edge devices. They put in place an compensation system, where edge devices that perform tasks for others will receive a reward. Every time a new task comes in at the BS, the appropriate node to perform it is computed based on several constraints (amount of computation power required, time to execute task among others), and after completion, receives payment. An auction system is put in place, where requesters (i.e. here users who send tasks to offload) put in bids to quantify their preference to offload to certain loads. While this approach presents a centralized network architecture as BSs allocate tasks, it presents a way to perform edge device to edge device offloading, even as those edge devices have divergent interests, through an efficient compensation system.

4.3 Prediction in edge computing

In [27], authors present a load balancing approach that predicts the load at an edge server in real time to decide whether or not a task should be offloaded. For that, they use a neural network with long short-term memory (LSTM), meaning neurons have a long lasting short term memory and can remember historical data. The model can predict which tasks are going to arrive at the server and what their requirements will be. When real tasks arrive, their characteristics are saved and added to the training samples. Then, a deep reinforcement learning model (DRL) receives the action predicted by the LSTM model, evaluates the error made by the model, and decides whether to follow the prediction, or perform a random action. Based on the rewards it receives, the DRL learns how to accurately act (meaning, to offload a task or perform it locally).

In [16], authors try to tackle the problem of efficient file caching in edge computing, to further reduce the time to provide service. To do so, next incoming tasks are predicted thanks to a Bayesian network. To do so, each time a file is accessed by an application, a relationship between the two is saved into the system. A task is then modelled as a combination of access to different application, and the problem of predicting which task the user is requesting becomes about computing the probability of said task knowing a given set of application is being accessed by the user. Authors consider tasks usually come in sets, and by performing statistics on which tasks usually follow others, they can compute a probable next task, and therefore find what should be considered as the logical next task.

4.4 Cloud and edge federation

Distributed data centers Interconnected, geographically distributed data centers are an interesting network configuration that deserves to be studied. Here, multiple data centers in different locations, interconnected over internet, work together to serve users. The authors of [24] give us an interesting load balancing approach based on a non cooperative game.
Here, a Cloud Workload Manager (CWM) is responsible for distributing the tasks coming in over all the data centers. Each data center has its Center Workload Manager, to distribute tasks over computing nodes. The CWM's job is to distribute workload, ensuring no task is dropped. The CWM also aims at limiting operation costs, such as energy costs and migrations over the network. Seeing this is an NP-hard problem, the authors model the workload distribution as a non cooperative game theory problem. Each data center will aim at reducing its own operation costs. They therefore propose a heuristic, the Nash Equilibrium based Intelligent Load Distribution (NILD) algorithm, to jointly optimize energy, network and delay costs. Experiments showed NILD brings operating costs down significantly. It also lowers task queuing times, and has a smaller runtime.
There are a lot of things we can learn from this paper. It takes into account a lot of different parameters for optimization. It also takes into account heterogeneity between centers. However, their delay model does not take into account the delay incurred by the migration over the network, only the delay incurred by queuing at arrival at the data center. Also, the model for computing power at a datacenter takes into account the

number of nodes and the number of each node's cores, making it a suitable model for our application as it would allow us to accurately model nodes constrained in computing power. However, the approach was only tested on data centers of significant size (over 4000 nodes). This means there is no data on whether this approach retains its efficiency with smaller, less powerful data centers.

In [4], authors explored a similar scenario, where a group of data centers at the network edge receive requests. If a given data center is saturated, it will forward requests to another neighboring data center with a probability p_1. Therefore, we are dealing with a cooperative approach here. They first model interactions between two data centers with a Markov chain. They then extend their model to N data centers. They calculate their model's performance by considering a scenario where two data centers experience the same arrival rate but one has a drastically smaller capacity than the other. In this case, the saturated data center experiences a notable decrease in its blocking probability, whereas the other data center observes little to no increase in its blocking probability. It is worth noting that this approach has not been tested on real machines, only via simulations. It also does not take into account the potential delay, cost and energy consumption incurred by transferring forwarding these requests to the other data center.

4.5 Data storage and replication

Data storage and replication is an important aspect of edge computing. In a data center environment, data is stored within the facility. In an edge architecture, as nodes are constrained in resources - nodes do not all store all data available in the system. However, it is mandatory for nodes to be able to quickly access the data they need, in order to serve users under a certain time constraint. Besides, having several copies of data can allow for smart replica placement where data is needed, but also serves as a backup in case data is lost. Therefore, a clever data storage and replication scheme needs to be in place to ensure this.

In [12], Xie et al present a strategy to efficiently distribute files in a network of edge nodes interconnected by software defined network (SDN) enabled switches. The aim is to evenly distribute the dataload over all nodes, and make the routing of the data more efficient. For this, the Greedy Routing for Edge Data (GRED) algorithm is introduced.

The first step of this algorithm is to compute the positions of all switches in a 2D euclidean space. For that, the M-position algorithm computes coordinates for each switch based off the matrix of shortest paths. The 2D space is then partitioned into nearly even regions. Placing data is then done by hashing the data identifier with SHA-256 and computing coordinates with the last 8 bytes of the hash. Then, these coordinates are placed on the 2D plane and whichever region they fall into determines the switch where data should be forwarded to. Data is then greedily forwarded there. This ensures data is spread out evenly among edge nodes. However, to make up for the fact that nodes might be heterogeneous in memory availability, edge nodes that are saturated can also change SDN control commands to re-route data to other edge nodes if needed.

Performance evaluations show this approach routes efficiently, and evenly distributed data over the network, even with a low number of edge servers in the network, and even as the amount of data increases.

In [7], authors present a decentralized, dynamic replica placement algorithm. They model the placement of replicas as a Facility Location Problem. This is a problem where facilities have to be placed in a given space in order to minimize distance to all customers demanding certain products, and is a typical issue in industrial engineering. In the case of edge computing, replicas play the role of facilities, users play the role of customers, demand is computed using the number of times a replica has been requested, and distance is the delay to receive the replica. The algorithm to replicate data is fairly straightforward : if a node requests data a certain amount of times and it causes too much overhead to send it, it will receive a copy of said data. Making replica management decentralized requires communication between nodes; but it shouldn't be too frequent, at the risk of causing unnecessary overhead. Therefore, whenever a new replica is created, only nodes susceptible to request this data in the near future are notified. This approach is decentralized and allows for data replication. However, it does not account for data modification; replicas are stable, and if a node was to update it, there would exist two versions of the replica that would still be labeled as the same. Besides, this approach only tackles data replication for increased availability, and not for back up.

It is worth repeating that these approaches aim at optimizing data placement for efficient recovery and reduce latency, whether it is by utilizing SDN possibilities for [12] or to place data closer to where it is needed in [7]. However, they do not account for replicas to backup data. In [22], authors present an approach at disaster recovery of data in an edge computing architecture. Given positions of edge nodes and predictive data for natural disasters, they figure out which nodes are in a dangerous zone and which aren't. Taking into account which safe nodes have the most available storage, data is forwarded to them. This is an approach at data backup in the event of a punctual and predictable disaster event, which is definitely a case that should be taken into account. However, loss of data does not only happen in the case of a disaster, and disasters are often not predictable. Therefore, such a system would be a good addition to our system, but it would not systematically prevent data loss.

5 Discussion

We've now roughly presented how data center consolidation works, then how peer to peer systems are similar to edge computing and how they tackle several common issues, and the current state of the art in peer to peer virtualization, load balancing and data replication. We will now discuss potential fields that might require further research.

Virtualization happens through virtual machines, containers and unikernels. Each has its pros and cons. Virtual machines cause more performance overhead, but they offer isolation and are arguably the most mature technology. Containers offer better performance and are faster to start, and it is fair to say the technology is also mature and widely used, but containers offer poor isolation by design. Unikernels offer great performance and isolation, but aren't nearly as widely used as the other two. Overall, using containers seems to be the most suitable for an edge computing architecture. If isolation is a must, containers and virtual machines can be coupled. It is also fair to say that unikernels will definitely play a role in edge computing, perhaps coupled with containers [9] , but this is not the standard yet.

As far as load balancing goes, it seems that optimization algorithms of random search to find optimal mapping of services and application have been extensively explored in data center architectures. These approaches are efficient meta heuristics that can approximate a solution to the NP hard problem of mapping virtual environments on the right amount of physical machines. When translating to an edge-fog layer, these methods have been adapted to respond to new constraints, bringing in game theory principles or reinforcement learning in certain cases. It is worth noting that most approaches consider all nodes belong to a single provider and can therefore be federated with a centralized network controller, and that the edge and fog layers are only antennas to a more centralized cloud. If we consider the possibility of an independent edge computing layer, decentralized load balancing would be a necessity, and therefore represents a potentially important research field.

We've also looked at prediction approaches, which undeniably make load balancing more efficient. These approaches make use of deep learning methods, and couple supervised learning and reinforcement learning. However, this means a large amount of data has to be available for initial training of models, and while these datasets might exist for data centers, they might not for edge computing. This is because edge computing is a newer paradigm, that is yet to be widely deployed, but also because data collected on a given location at the edge might not be representative of activity at a different location at the edge; especially given the small number of users served by a group of edge nodes compared to the number of users served by a data center. Auhors in [27] solves this issue by using data logged at the edge server and slowly building up the training data set as tasks come in, but this means predictions will likely be inaccurate until a sufficient amount of data is collected. Prediction approaches are therefore also a potential hot topic to investigate, as this would make load balancing even more efficient.

As far as data replication goes, having several copies of a same piece of data can happen for one of two reasons. The first one is in the case of a popular, highly requested data block. Replicating and disseminating data is done with the objective of making it more available to users who want it. Most approaches we have talked about here replicate for this reason, whether it is data centers replicating data on other data centers, edge computing servers disseminating data while optimizing retrieval time, or peer to peer systems that replicate popular data on the network. The second reason why data could be replicated is for backup purposes. There does not seem to be many approaches at a smart replication scheme oriented towards constant data backup. Most wide peer to peer networks have been oriented towards content distribution, so as we just said, replicating popular data. However, backed up data might not be popular, so its availability is not guaranteed. Modifications to these protocols are to be explored to ensure such a service.

Peer to peer systems also give us ways to deal with churn. Most edge computing papers we've examined do not address extensively address churn, because networks are federated through centralized controller, that can detect if a node has gone down. As far as data stored on the node, most approaches consider it is backed up on a remote cloud and will therefore not be lost. However, if we consider an independent, decentralized edge node network, these hypothesis do not stand, and churn should be worried about. Peer to peer systems resilient to churn cleverly adapt themselves as nodes go up and down, and make sure data is always stored at multiple locations and not lost. However, this usually relies on nodes communicating efficiently and reliably, which might not always be the case over the internet. Moreover, edge cluster might have a lower number of nodes available, meaning the failure of a single node affects the network in a harsher way.

Most recent peer to peer work is oriented towards cryptocurrencies, blockchain and smart energy grids. These approaches provide incentivization solutions, which could be a crucial part of enabling cooperation between edge nodes administrated by different providers, as storage and computing have a large energy cost. The idea of network credit, or payment through cryptocurrencies, have been explored but need further investigation.

6 Conclusion

In this paper, we looked at the current state of the art for creating data center like service on a decentralized network of federated, restricted edge nodes interconnected via the internet. First, we've looked at what a data center can currently offer and how this is achieved. Specifically, we've explored virtualization, load balancing, prediction and data replication. Then, we have taken a look at how peer to peer systems achieve decentralized federation of peers as well as efficient data replication and retrieval. Finally, we explored the current state of the art on edge computing, this time exploring virtualization on restricted nodes, load balancing and prediction under new constraints incurred by the edge network (unreliability of the network and availability of data), as well efficient data replication.

References

[1] Fabian Kuhn, Stefan Schmid, and Roger Wattenhofer. "Towards worst-case churn resistant peer-to-peer systems". en. In: *Distributed Computing* 22.4 (May 2010), pp. 249–267. ISSN: 0178-2770, 1432-0452. DOI: `10.1007/s00446-010-0099-z`. URL: `http://link.springer.com/10.1007/s00446-010-0099-z` (visited on 01/17/2023).

[2] Juan Benet. *IPFS - Content Addressed, Versioned, P2P File System*. arXiv:1407.3561 [cs]. July 2014. URL: `http://arxiv.org/abs/1407.3561` (visited on 01/03/2023).

[3] Waltenegus Dargie. "Estimation of the cost of VM migration". In: *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*. China: IEEE, Aug. 2014, pp. 1–8. ISBN: 978-1-4799-3572-7. DOI: `10.1109/ICCCN.2014.6911756`. URL: `http://ieeexplore.ieee.org/document/6911756/` (visited on 11/07/2022).

[4] Christine Fricker et al. "Analysis of an Offloading Scheme for Data Centers in the Framework of Fog Computing". en. In: *ACM Transactions on Modeling and Performance Evaluation of Computing Systems* 1.4 (Sept. 2016), pp. 1–18. ISSN: 2376-3639, 2376-3647. DOI: `10.1145/2950047`. URL: `https://dl.acm.org/doi/10.1145/2950047` (visited on 11/01/2022).

[5] Deze Zeng et al. "Joint Optimization of Task Scheduling and Image Placement in Fog Computing Supported Software-Defined Embedded System". In: *IEEE Transactions on Computers* 65.12 (Dec. 2016), pp. 3702–3712. ISSN: 0018-9340. DOI: `10.1109/TC.2016.2536019`. URL: `http://ieeexplore.ieee.org/document/7422054/` (visited on 11/01/2022).

[6] Shripad Nadgowda et al. "Voyager: Complete Container State Migration". In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. Atlanta, GA, USA: IEEE, June 2017, pp. 2137–2142. ISBN: 978-1-5386-1792-2. DOI: `10.1109/ICDCS.2017.91`. URL: `http://ieeexplore.ieee.org/document/7980161/` (visited on 11/07/2022).

[7] Atakan Aral and Tolga Ovatman. "A Decentralized Replica Placement Algorithm for Edge Computing". In: *IEEE Transactions on Network and Service Management* 15.2 (June 2018), pp. 516–529. ISSN: 1932-4537. DOI: `10.1109/TNSM.2017.2788945`. URL: `https://ieeexplore.ieee.org/document/8244318/` (visited on 01/01/2023).

[8] Antonio Marotta, Stefano Avallone, and Andreas Kassler. "A Joint Power Efficient Server and Network Consolidation approach for virtualized data centers". en. In: *Computer Networks* 130 (Jan. 2018), pp. 65–80. ISSN: 13891286. DOI: `10.1016/j.comnet.2017.11.003`. URL: `https://linkinghub.elsevier.com/retrieve/pii/S1389128617304012` (visited on 11/07/2022).

[9] Roberto Morabito et al. "Consolidate IoT Edge Computing with Lightweight Virtualization". In: *IEEE Network* 32.1 (Jan. 2018), pp. 102–111. ISSN: 0890-8044, 1558-156X. DOI: `10.1109/MNET.2018.1700175`. URL: `https://ieeexplore.ieee.org/document/8270640/` (visited on 12/30/2022).

[10] Vasileios Karagiannis et al. “Edge computing with peer to peer interactions: use cases and impact”. en. In: *Proceedings of the Workshop on Fog Computing and the IoT*. Montreal Quebec Canada: ACM, Apr. 2019, pp. 46–50. ISBN: 978-1-4503-6698-4. DOI: 10.1145/3313150.3313226. URL: https://dl.acm.org/doi/10.1145/3313150.3313226 (visited on 01/17/2023).

[11] Ilias Mavridis and Helen Karatza. “Combining containers and virtual machines to enhance isolation and extend functionality on cloud computing”. en. In: *Future Generation Computer Systems* 94 (May 2019), pp. 674–696. ISSN: 0167739X. DOI: 10.1016/j.future.2018.12.035. URL: https://linkinghub.elsevier.com/retrieve/pii/S0167739X18305764 (visited on 11/07/2022).

[12] Junjie Xie et al. “Efficient Data Placement and Retrieval Services in Edge Computing”. In: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. Dallas, TX, USA: IEEE, July 2019, pp. 1029–1039. ISBN: 978-1-72812-519-0. DOI: 10.1109/ICDCS.2019.00106. URL: https://ieeexplore.ieee.org/document/8885191/ (visited on 12/04/2022).

[13] Gala Yadgar et al. “Modeling The Edge:{Peer-to-Peer} Reincarnated”. In: *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*. 2019.

[14] Zheyi Chen et al. “Towards Accurate Prediction for High-Dimensional and Highly-Variable Cloud Workloads with Deep Learning”. In: *IEEE Transactions on Parallel and Distributed Systems* 31.4 (Apr. 2020), pp. 923–934. ISSN: 1045-9219, 1558-2183, 2161-9883. DOI: 10.1109/TPDS.2019.2953745. URL: https://ieeexplore.ieee.org/document/8902077/ (visited on 01/17/2023).

[15] Rabiya Khalid et al. “A Blockchain-Based Load Balancing in Decentralized Hybrid P2P Energy Trading Market in Smart Grid”. In: *IEEE Access* 8 (2020), pp. 47047–47062. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2979051. URL: https://ieeexplore.ieee.org/document/9026929/ (visited on 01/03/2023).

[16] Chunlin Li et al. “Adaptive priority-based cache replacement and prediction-based cache prefetching in edge computing environment”. en. In: *Journal of Network and Computer Applications* 165 (Sept. 2020), p. 102715. ISSN: 10848045. DOI: 10.1016/j.jnca.2020.102715. URL: https://linkinghub.elsevier.com/retrieve/pii/S1084804520301892 (visited on 01/17/2023).

[17] Gang Li and Jun Cai. “An Online Incentive Mechanism for Collaborative Task Offloading in Mobile Edge Computing”. In: *IEEE Transactions on Wireless Communications* 19.1 (Jan. 2020), pp. 624–636. ISSN: 1536-1276, 1558-2248. DOI: 10.1109/TWC.2019.2947046. URL: https://ieeexplore.ieee.org/document/8876867/ (visited on 01/17/2023).

[18] Guangshun Li et al. “A new load balancing strategy by task allocation in edge computing based on intermediary nodes”. en. In: *EURASIP Journal on Wireless Communications and Networking* 2020.1 (Dec. 2020), p. 3. ISSN: 1687-1499. DOI: 10.1186/s13638-019-1624-9. URL: https://jwcn-eurasipjournals.springeropen.com/articles/10.1186/s13638-019-1624-9 (visited on 12/31/2022).

[19] Amit M Potdar et al. “Performance Evaluation of Docker Container and Virtual Machine”. en. In: *Procedia Computer Science* 171 (2020), pp. 1419–1428. ISSN: 18770509. DOI: 10.1016/j.procs.2020.04.152. URL: https://linkinghub.elsevier.com/retrieve/pii/S1877050920311315 (visited on 12/27/2022).

[20] Meet Shah et al. “Decentralized Cloud Storage Using Blockchain”. In: *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)*. Tirunelveli, India: IEEE, June 2020, pp. 384–389. ISBN: 978-1-72815-518-0. DOI: 10.1109/ICOEI48184.2020.9143004. URL: https://ieeexplore.ieee.org/document/9143004/ (visited on 01/03/2023).

[21] Fatma M. Talaat et al. “A load balancing and optimization strategy (LBOS) using reinforcement learning in fog computing environment”. en. In: *Journal of Ambient Intelligence and Humanized Computing* 11.11 (Nov. 2020), pp. 4951–4966. ISSN: 1868-5137, 1868-5145. DOI: 10.1007/s12652-020-01768-8. URL: http://link.springer.com/10.1007/s12652-020-01768-8 (visited on 01/01/2023).

[22] Takuma Tsubaki et al. “Effective disaster recovery for edge computing against large-scale natural disasters”. In: *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC)*. Las Vegas, NV, USA: IEEE, Jan. 2020, pp. 1–2. ISBN: 978-1-72813-893-0. DOI: 10.1109/CCNC46108.2020.9045528. URL: https://ieeexplore.ieee.org/document/9045528/ (visited on 01/17/2023).

[23] Ahmed Awad et al. “A Novel Intelligent Approach for Dynamic Data Replication in Cloud Environment”. In: *IEEE Access* 9 (2021), pp. 40240–40254. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3064917. URL: https://ieeexplore.ieee.org/document/9373303/ (visited on 12/29/2022).

[24] Ninad Hogade, Sudeep Pasricha, and Howard Jay Siegel. “Energy and Network Aware Workload Management for Geographically Distributed Data Centers”. In: (2021). Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.2106.00066. URL: https://arxiv.org/abs/2106.00066 (visited on 12/04/2022).

[25] Fenghui Zhang and Michael Mao Wang. “Stochastic Congestion Game for Load Balancing in Mobile-Edge Computing”. In: *IEEE Internet of Things Journal* 8.2 (Jan. 2021), pp. 778–790. ISSN: 2327-4662, 2372-2541. DOI: 10.1109/JIOT.2020.3008009. URL: https://ieeexplore.ieee.org/document/9136735/ (visited on 01/01/2023).

[26] Ali Majed, Fatemeh Raji, and Ali Miri. “Replication management in peer-to-peer cloud storage systems”. en. In: *Cluster Computing* 25.1 (Feb. 2022), pp. 401–416. ISSN: 1386-7857, 1573-7543. DOI: 10.1007/s10586-021-03395-0. URL: https://link.springer.com/10.1007/s10586-021-03395-0 (visited on 01/17/2023).

[27] Youpeng Tu et al. “Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning for Efficient Edge Computing in IoT”. en. In: *Future Internet* 14.2 (Jan. 2022), p. 30. ISSN: 1999-5903. DOI: 10.3390/fi14020030. URL: https://www.mdpi.com/1999-5903/14/2/30 (visited on 01/17/2023).