Nuno Morais

Master of Science

MicromanEdge: a monitoring and orchestration protocol for microservices in Edge Environments

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

Orientador:	João Leitão, Assistant Professor, NOVA University of Lisbon

Júri

Presidente:	Name of the committee chairperson
Arguente:	Name of a raporteur
Vogal:	Yet another member of the committee

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

December, 2019

Resumo

Lorem ipsum em Português.

Palavras-chave: Palavras-chave (em Português) ...

Abstract

Lorem ipsum in english.

Keywords: Keywords (in English) . . .

Índice

Capítulo 1

Introduction

1.1 Context

Nowadays, the Cloud Computing paradigm is the standard for development, deployment and management of services, it has proven to have massive economic benefits that make it very likely to remain permanent in future of the computing landscape. It provides the illusion of unlimited resources available to services, and has changed the way developers, users and businesses rationalize about applications [1].

Currently, most software present in our everyday life such as Google Apps, Amazon, Twitter, among many others is deployed on some form of cloud service. However, currently, the rise in popularity of mobile applications and IoT applications differs from the centralized model proposed by the Cloud Computing paradigm. With recent advances in the IoT industry, it is safe to assume that in the future almost all consumer electronics will play a role in producing and consuming data. As the number of devices at the edge and the data they produce increases rapidly, transporting the data to be processed in the Cloud will become unfeasible.

Systems that require real-time processing of data may not even be feasible with Cloud Computing. When the volume of data increases, transporting the data in real time to a Data Center is impossible, for example, a Boeing 787 will create around 5 gigabytes of data per second [8], and Google's self-driving car generates 1 Gigabyte every second [17], which is infeasible to transport to the DC for processing and responding in real-time.

When all computations reside in the data center (DC), far from the source of the data, problems arise: from the physical space needed to contain all the infrastructure, the increasing amount of bandwidth needed to support the information exchange from the DC to the client, the latency in communication from the client to the DC as well as the security aspects that arise from offloading data storage and computation, have directed us into a post-cloud era where a new computing paradigm emerged, Edge Computing.

Edge computing takes into consideration all the computing and network resources that act as an "edge"along the path between the data source and the DC and addresses the increasing need for supporting interaction between cloud computing systems and mobile or IoT applications [18]. However, when accounting for all the devices that are external to the DC, we are met by a huge increase in heterogeneity of devices: from Data Centers to private servers, desktops and mobile devices to 5G towers and ISP servers, among others.

1.2 Motivation

The aforementioned heterogeneity implies that there is a broad spectrum of computational, storage and networking capabilities along the edge of the network that can be leveraged upon to perform computations that rely on the individual characteristics of the devices performing the tasks, which can vary from from generic computations to aggregation, summarization, and filtering of data. [11]

There have been efforts to move computation towards the Edge of the network, Fog Computing [20], which is an extension of cloud computing from the core of the network to the edge of the network, has shown to benefit web application performance [23]. Additionally, Content Distribution Networks [] and Cloudlets [] are an extension of this paradigm and are extensibly used nowadays.

Recently, and hand in hand with the trend of cloud decentralization, applications have strayed away from their monolithic application architecture towards splitting in smaller, cohesive, and independent services (microservices). Microservices interact using messages [6], and may cooperate to perform applicational tasks. Specializing services provides many advantages for businesses: each service may be independently built, versioned, and maintained, in addition to the load-balancing, scalability and fault tolerance that arise from employing a decentralized architecture [14].

However, the study state of the art reveals that there is a lack of unified infrastructure that provides the necessary scalability along with resource location and monitoring in to efficiently support dynamic service deployment and management in the Edge. An infrastructure capable of performing the aforementioned tasks successfully has strong applicability in todays world: from commercial purposes (for example, Cloud providers that look to better employ IoT devices), or Smart Cities/Countries.

Given this, p2p protocols must be devised that provide exact resource location and monitoring while providing resource locality according the device distribution in the Edge environment. A common approach to leverage on device heterogeneity is to federate devices in hierarchical topologies. Hierarchical topologies may be designed to handle the network and device instability that arises when relying on devices that do not have the same infrastructure as those in Data Centers. Consequently, the hierarchy be used to employ efficient aggregation protocols towards device and service monitoring.

1.3 Expected Contributions

The expected contributions from this work are two fold:

- Devise a framework built on top of a membership algorithm that employs a hierarchical topology which reflects the Edge device distribution. Then, exploit the topology to perform monitoring over device and service status. Finally, aggregate those values and employ them towards efficiently deploying services in Edge environments.
- Test the topology in well-known simulators to ensure its correctness and to compare performance with popular systems.
- Design an experimental scenario that permits direct comparison with other service deployment systems under various scenarios, these will be performed by tracking application performance and overlay metrics while varying network instability, device heterogeneity, and service load.
- paragrafo sobre como usar o

1.4 Thesis structure

This document is structured in the following manner:

Chapter 2 focuses on the related work, first and second sections cover P2P systems and the different types of topology management protocols, with an emphasis on structured and self-adapting overlays, third section studies the different types of aggregation and popular implementations for each aggregation type. Fourth section addresses resource discovery and how to perform efficient searches over networks composed by a large number of devices. Finally, last section discusses recent approaches towards enabling Edge Computing along with discussion about Fog, Mist and Osmotic Computing.

Chapter 3 further explains the proposed contribution along with the work plan for the remainder of the thesis.

Capítulo 2

Related Work

The following chapter provides context about the challenge we attempt to solve and presents the studied related work towards the identified sub-challenges in deploying and managing microservices in the edge of the network.

First, **microservice networking**: microservices need to cooperate towards solving applicational tasks, as such, microservices need to be integrated in an efficient abstraction layer that allows them to find each other and communicate. This raises an old challenge in P2P computing: how can peers organize themselves in the network such that they can find resources (e.g. other peers, services or even computing power) in an efficient way? It is important to notice that the edge environment is composed by lots of sub-networks of heterogenous devices concurrently entering and leaving the network, which presents a hard scenario in **resource location systems**, particularly in creating and maintaining the topology (**topology management**), because the overlay needs to federate large numbers of devices and adapt to the underlying network in order to remain efficient.

Addressing this, section 2.1 of this chapter provides context about **topology management**, particularly the main categories of topologies and discuss their applicability in edge environments. Second section, 2.2 covers **resource location architectures**, which leverage on one (or a combination of) topologies to index resources in the system. For each type of architecture we discuss their differences and present popular implementations in the state of the art.

Now, provided that we have an efficient overlay tailored towards deploying microservices, the next step is **ensuring and maintaining quality of service**. For this, we need to perform **monitoring** of device and service status, with a special emphasis on devices in the edge environment. Section 2.3 covers popular techniques used towards tracking device and service status, we particularly study failure detectors and study metrics assist in pinpointing causes of QoS degradation. Furthermore, section 2.3 also covers related work about monitoring edge devices and some examples of popular monitoring systems.

However, if we wish to gather metrics on device and service status, devices will produce a continuous stream of monitoring data, transmitting and storing that data becomes a limitation to the scalability of the system. To circumvent this, that data needs to somehow be combined, this process is called **aggregation**. Section 2.4 studies aggregation, which consists in the process of combining several numeric values into one single representative [0]. Next, we discuss the different types of aggregation, and how they can be applied towards maintaining the aforementioned quality of service.

Lastly, section 2.5 studies how the aggregated results can be used to perform microservice management and deployment. Namely, how to orchestrate microservice deployment and migration such that the computation is offloaded to the Edge. We discuss the difference between paradigms such as Fog Computing, Osmotic Computing and Edge Computing. Lastly, we discuss approaches towards implementing elastic computing in Edge environments.

2.1 Topology Management

2.1.1 P2P systems

As previously mentioned, a challenge towards solving the proposed solution is to federate all peers in an abstraction layer that allows intercommunication and efficient resource discovery. Given that this is a classic P2P problem, this section provides context about P2P systems and how they usually organize.

In P2P, participants contribute to the system with a portion of their resources,so that that the overall system can accomplish tasks that would otherwise be impossible for a single peer to solve. However, due to memory and communication overhead, it is undesirable that all nodes in a P2P system collaborate with all other peers (unless in specific scenarios which we will further elaborate).

Peers select a subset of peers in the system to establish neighboring relations. These neighboring relations are usually constructed on top links from an already existing network (commonly called an underlay). The accumulation of neighboring relations on top of the underlay network is what constitutes the **overlay network**. Overlay networks are commonly categorized in four categories: structured, unstructured, flat, and hierarchical:

2.1.2 Unstructured overlays

Unstructured overlays usually impose little to no rules in neighboring relations, peers may pick random peers to be their neighbors, or alternatively employ strategies to "rank"neighbors and selectively pick the "best". A key factor of unstructured overlays is that nodes can easily replace failed neighbors. Which provides a low maintenance overhead and provides high resilience to participants concurrently entering and leaving the system (this is called churn [0]). Unstructured overlays present an attractive option to federate edge devices, as edge devices do not need use many resources to join and participate in the system, and can easily adapt to the dynamic environment.

Scamp

Hyparview

Overnesia

2.1.3 Structured overlays

Structured overlays enforce strong rules towards neighbor selection (generally based on identifiers of peers). As a result, the overlay generally converges to a topology known a priori, where the target topologies are normally tailored towards applicational requirements. A canonical example of a type of structured overlay is a distributed hash table (DHT), peers in a DHT use consistent hashing functions to select random identifiers that are uniformly distributed over the identifier space. Then, DHTs offer efficient routing capabilities over the identifier space (usually routing procedures take a logarithmic number of steps). DHTs have been extensibly used to support many large scale services (publish-subscribe, file sharing, among others) and are especially used in Cloud-based environments.

2.1.4 Hierarchical and flat overlays

Flat or hierarchical overlays Flat overlays are composed by peers that evenly share the tasks of the system, there is no differentiation between the resources that peers have to contribute to the system. Contrary their flat counterpart, hierarchical overlays may be characterized as overlays where peers have different tasks in the system according to their roles, this is an easy way to accommodate device heterogeneity while potentially increasing the performance of the system. Hierarchical structures can be applied to both structured and unstructured overlays: in the case of unstructured overlays, the canonical example is **super-peers**, which are peers that have increased capacity and stability, this is the approach taken by Gia [0] to improve the scalability of Gnutella [0]. Super Peers are commonly assigned with the task of disseminating queries throughout the system so that other peers do not have to. This technique effectively reduces the number of peers that have to exchange messages, which by consequence raises system scalability, however, this approach is still inefficient when finding rare resources in the system. The second approach to building a hierarchical topology is to employ a DHT, hierarchical DHTS usually form contained DHTS within other DHTS (e.g. a ring within a ring). This offers several important advantages over a flat DHT: first, lookups take less hops and messages to reach the target, second, organizing nodes in disjoint groups allows traffic locality if groups of nodes are close in the underlay, finally, churn events within a group stay contained within that group. However, many of these systems either employ more memory to accommodate the many levels hierarchical DHT, or tradeoff reliability (by shortening the number of connections) for memory and communication efficiency.

2.1.5 Evaluating topologies

Given that the accumulation of the neighboring relations among peers form a graph, one can define a set of metrics in order to measure direct properties of overlay topologies. Following, we list direct metrics to measure overlay performance:

1. **Connectivity**. A connected graph is one where there is at least one path from each node to all other nodes in the system. The absence of this property means that there are nodes in the system that are isolated, thus will not be able to cooperate towards the overall behavior of the system. This property is usually measured as a percentage, corresponding to the largest portion of the system that is connected. Intuitively, a connected overlay has 100% connectivity.

2. **Degree Distribution**. The degree of a node consists in the number of arcs that are connected to it. Depending on the type of system, the connections may be directed or indirected, in a directed graph there is a distinction between **in-degree** and **out-degree** of a node. Intuitively, nodes with a high in-degree have higher reachability in the system, and nodes with 0 in-degree cannot be reached. In flat overlays, where load distribution is desired, degree distribution should be as similar as possible in all nodes. By contrast, in hierarchical overlays, designs take advantage of device heterogeneity to differentiate between peers and promote scalability.

3. **Average Shortest Path**. A path is composed by the edges of the graph that a message would have to cross to get from one node to other. The average shortest path consists in the average of all those paths, to promote efficient communication patterns, is desirable that this value is as low as possible.

4. **Clustering Coefficient**. The clustering coefficient provides a measure of the density of neighboring relations across the neighbors of a given node. It consists in the number of a node's neighbors divided by the maximum number of links between those neighbors. Similar to the average shortest path, the clustering coefficient of an overlay consists in the average of the clustering coefficient of all the peers. A high value of clustering coefficients will result in a higher number of redundant messages, and by consequence, additional localized traffic. Finally, areas of an overlay with a higher clustering coefficient tend to be more easily isolated from the remaining system.

5. **Overlay Cost**. If we assume that a link in the overlay has a *cost*, then the overlay cost is the sum of all the links that form the overlay. Link cost can derive from overlay metrics (numeric distance, XOR distance, etc), or external metrics such as latency.

2.1.6 Discussion

2.2 Resource Location and Discovery

Given that the main challenge to solve is to provide a platform which enables microservice deployment and management in Edge devices, it is imperative that services are able to find the resources they need (either other services or peers in the system) to meet their requests. For this, peers need implement a **resource location system**.

Resource location systems are one of the most common applications of the P2P paradigm. In these systems, a participant provided with a resource descriptor is able to query peers and obtain an answer to the location (or absence) of that resource in the system within a reasonable amount of time. There are many types of queries a resource location system may support, we present some of them:

1. **Exact Match queries** specify the resource to search by the value of a specific attribute (for example, a hash of the value).
2. **Keyword queries** employ one or more keywords (or tags) combined with logical operators to describe resources (e.g. "pop", "rock", "pop and rock"...). These queries return a list of resources and peers that own a resource whose description matches the keyword(s).
3. **Range queries** retrieve all resources whose value is contained in a given interval (e.g. "movies with 100 to 300 minutes of duration"). These queries are especially applied in databases.
4. **Arbitrary queries** are queries that aim to find a set of nodes or resources that satisfy one or more arbitrary conditions, a possible example of an arbitrary query is looking for a set resources with a certain size or format.
5.

Disseminating the previously mentioned queries in an efficient manner through the overlay is a challenge in P2P resource location systems, there have been devised many **dissemination strategies** whose applicability depends on the applicational requirements and system capabilities. There are two main types of dissemination of queries, **flooding** and **walks**.

When **flooding**, peers eagerly forward queries to other peers in the system , the objective of flooding is to contact a certain number of distinct peers in the system that may have the desired resource. One approach is **complete flooding** which consists in contacting every node in the system, this guarantees that if the resource exists, it will be found (this is the only way to provide exact resource location in a decentralized resource location system), however, complete flooding is not scalable and incurs lots of message redundancy. **Flooding with limited horizon** minimizes the message redundancy overhead by attaching a TTL to messages that limits the number of times a message can be retransmitted. However, there is a trade-off for efficiency: flooding with limited horizon does not provide exact resource location. There are many other dissemination techniques, often tailored towards specific application requirements.

Walks are a dissemination strategy that attempts to minimize the communication overhead that accompanies flooding. Instead of peers forwarding messages to multiple peers, walks are forwarded one peer at a time throughout the system. How walks are propagated in the system dictates the type of walk being used: if walks are propagated randomly, they are said to be **random walks**.

Conversely, walks may take a biased paths in the system based on information accumulated by peers, this is called a **guided walk**. Guided walks implement the concept of routing indices that allow nodes to forward queries to neighbors that are more likely to have answers [0]. Another approach to bias walks is to use bloom filters [0], which is a space-efficient probabilistic data structure that supports set membership queries. Again, there are many techniques of performing guided walks, often tailored for specific system needs.

Throughout the years 3 popular architectures emerged that are common towards indexing resources in a distributed system:

2.2.1 Centralized Architectures

Centralized architectures rely on one (or a group of) centralized peers that index all resources in the system. This type of architecture greatly reduces the complexity of systems, as peers only need to contact a subset of nodes to locate resources. However, their scalability is limited, due to the centralized point of failure.

It is important to notice that in a centralized architecture, while the indexation of resources is centralized, the resource access may still be distributed (e.g. a centralized server provides the addresses of peers who have the files, and files are obtained in a pure P2P fashion). Some systems use a combination of architectures with success: file sharing systems like Napster and BitTorrent [0] are two examples of hybrid resource location systems that lasted the test of time.

Because centralized architectures have limited scalability, purely centralized architectures cannot be applied in large scale Edge environments. However, there are many ways that a hybrid architecture can be applied to Edge computing: since the failure rate of a single DC is low, if we assume a system composed by multiple DCs, they may act as a reliable failover for whenever edge devices are partitioned of fail. DCs can also act as an entrypoint for systems, or finally, act as an optimization reference point for peers.

2.2.2 Unstructured Architectures

2.2.3 Distributed Hash tables

Distributed Hash Tables (DHTs) contrast with centralized servers, where the index distribution is split among peers in the system. In a DHT, peers are assigned uniformly distributed IDs using hash functions, then, peers employ a global coordination mechanism that restricts their neighboring relations (usually called routing tables) such that the resulting overlays commonly consist in low-diameter geometric structures like rings, hypercubes, among others.

Peers maintain routing tables to forward messages in the system, such that they can contact any other participant in a bounded number of steps, where the bound (usually logarithmic) is dictated by the topology . Finally, using the same hash functions to map resources (files, multimedia, messages, etc.) to the peer identifier space, and assigning a key-space interval to each peer, peers can store and find any resource in a bounded number of steps (**exact resource location**).

One particular type of DHT that is commonly employed in in small to medium sized storage solutions is the One-Hop DHT, nodes in a one-hop DHT have **Full membership** of the system and consequently, can perform lookups in O(1) time and message complexity. Facebook's Cassandra [0] and Amazon's Dynamo [0] are widely used implementations of one-hop DHTs. However, full membership solutions have scalability problems due to the required memory and message volume necessary to maintain the full membership information up-to-date. Finally, maintaining full membership is costly in the presence churn (participants entering and leaving the system concurrently). The accumulation of these factors make full-membership solutions impractical in Edge environments.

Given this, the usual approach to building a DHT is through **partial membership** systems, which rely on some membership mechanism that restricts neighboring relations that are used to perform communication. DHTs with partial membership are attractive because they provide exact resource location while maintaining very little membership information (typically 1% of the peers), .

There have been attempts to apply DHTs towards Edge Computing. Common limitations that arise from this is that the common flat design that goes against the device heterogeneity of Edge Environments. Furthermore, given that devices in those environments have lower computational power and weaker connectivity, devices in the edge may even be a bottleneck to the system.

Following, we present some popular implementations of relevant DHT's along with a discussion on their applicability towards Edge environments:

Chord [19] is a distributed lookup protocol that addresses the need to locate the node that stores a particular data item, it specifies how to find the locations of keys, how nodes recover from failures, and how nodes join the system. Chord assigns each node and key an m-bit identifier that is uniformly distributed in the id space (peers receive roughly the same number of keys). Peers are ordered by identifier in a clockwise circle, then, any key *k* is assigned to the first peer whose identifier is equal or follows k in the identifier space.

Chord implements a system of "shortcuts" called the **finger table**. The finger table contains at most *m* entries, each *ith* entry of this table corresponds to the first peer that succeeds a certain peer *n* by 2^{ith} in the circle. This means that whenever the finger table is up-to-date, lookups only take logarithmic time to finish.

Chord, although provides the best trade-off between bandwidth and lookup latency [12], however, chord presents some limitations: peers do not learn routing information from incoming requests and links have no correlation to latency or traffic locality.

Chord is a basis for lots of work: Cyclone [2] is a hierarchical version of Chord provides that constructs a hierarchy by splitting the ID space into a PREFIX and SUFIX. The PREFIX provides intra-cluster identity, whereas the SUFIX is used towards creating clusters of nodes. Routing procedures are executed in lower rings and move up the hierarchy. Hieras [22] uses a binning scheme according the underlay topology to group peers into smaller rings. The lower the ring, the smaller the average link latency. Routing is similar to Cyclone. Crescendo [9] splits the ID range into domains (similar to DNS), where nodes in leaf-domains form Chord rings, then nodes merge rings by applying rules such that rings in different domains can communicate. The resulting routing table and the routing procedures in Crescendo are similar to chord.

Pastry [15] is a DHT that assigns a 128-bit node identifier (nodeId) to each peer in the system. The nodes are randomly generated thus uniformly distributed in the 128-bit nodeId space. Nodes store values whose keys are also distributed in the nodeId space. Key-value pairs are stored among nodes that are numerically closest to the key. This is accomplished by: in each routing step, messages are forwarded to nodes whose nodeId shares a prefix that is at least one bit closer to the key. If there are no nodes available, Pastry routes messages towards the numerically closest nodeId. This routing technique accomplishes routing in O(log N), where N is the number of Pastry nodes in the system. This protocol has been widely used and tested in applications such as Scribe [16] and PAST [7]. Limitations from using Pastry arise from the use of a numeric distance function towards the end of the routing process, which creates discontinuities at some node ID values, and complicates attempts at formal analysis of worst case behavior.

Kademlia [13] is a DHT with provable consistency and performance in a fault-prone environment. Kademlia nodes are assigned 160-bit identifiers uniformly distributed in the ID space. Peers route queries and locate nodes by employing a novel **XOR-based distance** function that is symmetric and unidirectional. Each node in Kademlia is a router whose routing tables consist of shortcuts to peers whose XOR distance is between 2^i by 2^{i+1} in the ID space. Intuitively, and similar to Pastry, "closer"nodes are those that share a longer common prefix. The main benefits that Kademlia draws from this approach are: nodes learn routing information from receiving messages, there is a single routing algorithm for the whole routing process (unlike Pastry) which eases formal analysis of worst-case behavior. Finally, Kademlia exploits the fact that node failures are inversely related to uptime by prioritizing nodes that are already present in the routing table.

Kelips [10] exploits increased memory usage and constant background communication to achieve O(1) lookup time and message complexity. Kelips nodes are split in k affinity groups split in the intervals $[0,k-1]$ of the ID space, thus, with n nodes in the system, each affinity group contains $\frac{n}{k}$ peers. Each node stores a partial set of nodes contained in the same affinity group and a small set of nodes lying in foreign affinity groups. Assuming a proportional number of files and peers in the system and a fixed view of nodes in foreign affinity groups, Kelips achieves O(1) time and message complexity in lookups at the cost of increased memory consumption (O($\sqrt{n}$). Due to this, system scalability is limited when compared to Pastry, Chord or Kademlia.

Tapestry [21] Is a DHT similar to pastry where messages are incrementally forwarded to the destination digit by digit (e.g. ***8 -> **98 -> *598 -> 4598). Lookups have logb(n) time complexity where b is the base of the ID space. A system with n nodes has a resulting topology composed of n spanning trees, where each node is the root of its own tree. Because nodes assume that the preceding digits all match the current node's suffix, it only needs to keep a constant size of entries at each route level. Thus, nodes contain entries for a fixed-sized neighbor map of size b.log(N).

2.2.4 Hybrid approaches

Curiata Build One Get One Free

2.2.5 Discussion

2.3 Monitoring

2.3.1 Device monitoring

2.3.2 Service monitoring

2.3.3 Relevant monitoring services

2.3.4 discussion

2.4 Aggregation

Aggregation is an essential building block on modern distributed systems, it enables the determination of important system wide properties in a decentralized manner [0]. Aggregation consists in computing an aggregation function over a set of input values where each node has one input value. Common aggregation functions consist in sum, count, average, min, max, where each function presents different properties which must be considered.

Towards deploying services in edge devices, aggregation may be used to monitor the device and service state (e.g. computing the average latency of the closest available service that meets a certain criteria; counting nearby available computing resources that can be

	Decomposable		Non-Decomposable
	Self-decomposable		
Duplicate insensitive	Min, Max	Range	Distinct Count
Duplicate sensitive	Sum, Count	Average	Median, Mode

Tabela 2.1: popular aggregation functions in function of decomposability and duplicate sensitiveness

used to offload services, or identify hotspots by aggregating the average system load in certain areas). Given this, it is important to understand the challenges of aggregating values in a highly decentralized manner. There are two properties of aggregation functions: *decomposable functions* and *duplicate sensitive* functions.

2.4.1 Properties of aggregation functions

For some aggregation functions, we may need to involve all elements in the multiset, however, for memory and bandwidth issues, it is impractical to perform a centralized computation, hence, the aim is to employ *in-transit computation*. In order to enable this, it is required that the aggregation function is **decomposable**. Intuitively, a decomposable aggregation function is one where a function may be composed defined as a composition of other functions. Decomposable functions may *self-decomposable*, which intuitively means that the aggregated value is the same for all possible combinations of all sub-multisets partitioned in the multiset. This happens whenever the applied function is commutative and associative (e.g. min, max, sum, count). A canonical example of a decomposable function that is not self-decomposable is average, which consists in the sum of all pairs divided by the count of peers that contributed to the aggregation.

The second property of aggregation is **duplicate sensitiveness**, and it is related to wether a given value occurs several times in a multiset. Depending on the aggregation function used, the presence of repeated values may influence the result, it is said that a function is **duplicate sensitive** if the result of the aggregation function is influenced by the repeated values (e.g. SUM), conversely, if the aggregation function is **duplicate insensitive** it can be successfully repeated any number of times to the same multiset without affecting the result (e.g. MIN and MAX).

Table 2.1 classifies popular aggregation functions in function of decomposability and duplicate sensitiveness as found in [0]:

Building on the concepts of duplicate sensitiveness and decomposability, we show that aggregation functions present their own particularities which dictate their applicability in particular scenarios. For example, a Min or Max function may be easier to implement with a simpler algorithm, while Sum, Count and Average require extra considerations. This presents a limitation towards calculating exact aggregations in large scale systems, to circumvent this, some systems do not require obtaining exact aggregated values to perform near optimally (e.g. estimating the system size in order to select the optimal fanout for a gossip system only requires an estimation of the magnitude of the system).

2.4.2 Aggregation techniques

Following, we present the studied categories of aggregation techniques: Hierarchical, Averaging, Sketches (hash or min-k based), Digests, Deterministic and Sampling. In each technique, we discuss its applicability in the edge environment.

2.4.2.1 Hierarchical

Hierarchical approaches leverage directly on the decomposability of aggregation functions. Aggregations from this class depend on the existence of a hierarchical communication structure, (e.g. a spanning tree) with one root (sink node). Aggregations take place by splitting inputs into groups and aggregating values bottom-up in the hierarchy. Commonly, hierarchical aggregation systems have nodes whose roles are *aggregators* or *forwarders*, intuitively, aggregators compute the aggregation functions forward results to forwarders who transfer results to upper levels in the hierarchy. In the absence of faults, the correct final result is obtained in the sink node. Many systems employ hierarchical approaches to aggregation, namely TAG [], DAG [], among others. Hierarchical approaches, due to taking advantage of device heterogeneity, are attractive in edge environments. However, due to the low computational power of devices, not all nodes may be able to handle the additional overhead of maintaining the hierarchical topology.

Averaging aggregation consists in the continuous computation and exchanging of partial averages data among all active nodes in the aggregation process. In this type of systems, after a few rounds, all nodes usually converge to the correct value with high accuracy, as shown in [0]. This type of aggregation is attractive for gossip protocols, where nodes may employ varied gossip techniques to continuously share and update their values with random neighbors. Algorithms from this category are also attractive to use in edge environments, because they are accurate while employing random unstructured overlays, which retain their fault-tolerance and resilience to churn.

Sketches are fixed-size data structures that hold a *sketch* of all network values. Multiple sketches are usually forwarded throughout the system, and nodes who forward sketches apply (usually commutative and associative) operations to update and merge them. functioning and edge discussion

Digests are an aggregation technique that gathers a representation of all system values, it supports complex aggregation functions such as Median and Mode. In short, algorithms employ a fixed-size data structures commonly composed of a set of values and associated counters) which compacts the data distribution (e.g. into a histogram). edge discussion

Counting algorithms target the same aggregation function: Count, algorithms from this class usually employ some randomized procedure to achieve a probabilistic approximation of the population size.

2.4.3 Relevant aggregation protocols

In this subsection we will analyze relevant aggregation protocolsthat ilustrate some techniques discussed above.

TAG: Tiny AGgregation[0] is a service for aggregation in low-power, distributed, wireless sensor networks. TAG distributes queries in the network in a time and power-efficient manner by employing a hierarchical aggregation pattern. For each aggregation procedure, there is a *root* nodes which broadcasts a message to start the tree-building process, each message contains two fields: a level and a an ID. Whenever a node without an assigned level receives a tree-building message, it assigns its own level as the message level plus one, and its own parent as the message sender. Then, it reassigns the level and ID to its own and forwards the message to other nodes. Then, whenever a node wishes to send a message to the root, it simply forwards the message bottom-up in the tree. The formed topology allows the computation of Count, Maximum, Minimum, Sum and Average. It is important to notice that the formed tree will be unbalanced as a function of the underlay latency and processing time.

DECA [0] //TODO

Astrolabe [0] //TODO

SingleTree [] **and MultipleTree** [] //TODO

2.4.4 Discussion

2.5 Offloading computation to the Edge

2.5.0.1 Decentralizing clouds

2.5.0.2 Fog Computing

2.5.0.3 Edge Computing

2.5.0.4 Osmotic Computing

Capítulo 3

Proposed Solution

To achieve this, we propose to create a new novel algorithm which employs a hierarchical topology that resembles the device distribution of the Edge Infrastructure. This topology is created by assigning a level to each device and leveraging on gossip mechanisms to build a structure resembling a FAT-tree [].

The levels of the tree will be determined by ...undecided... and will the tree be used to employ efficient aggregation and search algorithms. Each level of the tree will be composed by many devices that form groups among themselves, the topology of the groups ...undecided...

The purpose of this algorithm is to allow:

1. Efficient resource monitoring to deploy services on.
2. Offloading computation from the cloud to the Edge and vice-versa through elastic management of deployed services.
3. Service discovery enabled by efficiently searching over large amount of devices
4. Federate large amount of heterogeneous devices and use heterogeneity as an advantage for building the topology.

We plan to research existing protocols (both for topology management and aggregation) and enumerate their trade-offs along with how they behave across different environments. Then, employ a combination of different techniques according to their strengths in a unique way that is tailored for this topology.

3.1 Document Structure

Capítulo 4

Planning

4.1 Proposed solution

4.2 Scheduling

Bibliografia

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica e et al. "A View of Cloud Computing". Em: *Commun. ACM* 53.4 (abr. de 2010), 50–58. ISSN: 0001-0782. DOI: 10.1145/1721654.1721672. URL: https://doi.org/10.1145/1721654.1721672.

[0] M. S. Artigas, P. García e A. F. Skarmeta. "DECA: A hierarchical framework for DECentralized aggregation in DHTs". Em: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 4269 LNCS (2006), pp. 246–257. ISSN: 16113349. DOI: 10.1007/11907466_23.

[2] M. S. Artigas, P. G. López, J. P. Ahulló e A. F. Skarmeta. "Cyclone: A novel design schema for hierarchical DHTs". Em: *Proceedings - Fifth IEEE International Conference on Peer-to-Peer Computing, P2P 2005*. Vol. 2005. 2005, pp. 49–56. ISBN: 0769523765. DOI: 10.1109/P2P.2005.5.

[0] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham e S. Shenker. "Making Gnutella-like P2P Systems Scalable". Em: *Computer Communication Review* 33.4 (2003), pp. 407–418. ISSN: 01464833. DOI: 10.1145/863997.864000.

[0] B. Cohen. "Incentives build robustness in BitTorrent". Em: *Workshop on Economics of Peer-to-Peer systems*. Vol. 6. 2003, pp. 68–72.

[0] A. Crespo e H. Garcia-Molina. "Routing indices for peer-to-peer systems". Em: *Proceedings 22nd International Conference on Distributed Computing Systems*. 2002, pp. 23–32. DOI: 10.1109/ICDCS.2002.1022239.

[0] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall e W. Vogels. "Dynamo: amazon's highly available key-value store". Em: *ACM SIGOPS operating systems review*. Vol. 41. 6. ACM. 2007, pp. 205–220.

[6] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin e L. Safina. "Microservices : Yesterday , Today , and Tomorrow". Em: (), pp. 195–196.

[7] P. Druschel e A. Rowstron. "PAST: a large-scale, persistent peer-to-peer storage utility". Em: *Proceedings Eighth Workshop on Hot Topics in Operating Systems*. 2001, pp. 75–80. DOI: 10.1109/HOTOS.2001.990064.

[8] M. Finnegan. *Boeing 787s to create half a terabyte of data per flight, says Virgin Atlantic*. 2013. URL: https://www.computerworld.com/article/3417915/boeing-787s-to-create-half-a-terabyte-of-data-per-flight--says-virgin-atlantic.html.

[9] P. Ganesan, K. Gummadi e H. Garcia-Molina. "Canon in G major: Designing DHTs with hierarchical structure". Em: *Proceedings - International Conference on Distributed Computing Systems* 24 (2004), pp. 263–272. DOI: 10.1109/icdcs.2004.1281591.

[0] M. Grabisch, J.-L. Marichal, R. Mesiar e E. Pap. *Aggregation functions*. Vol. 127. Cambridge University Press, 2009.

[0] *Gtk-Gnutella*. 2019. URL: https://sourceforge.net/projects/gtk-gnutella/.

[10] I. Gupta, K. Birman, P. Linga, A. Demers e R. Van Renesse. "Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead". Em: *International Workshop on Peer-to-Peer Systems*. Springer. 2003, pp. 160–169.

[0] M. Jelasity, A. Montresor e O. Babaoglu. "Gossip-Based Aggregation in Large Dynamic Networks". Em: *ACM Transactions on Computer Systems* 23 (ago. de 2005), pp. 219–252. DOI: 10.1145/1082469.1082470.

[0] P. Jesus, C. Baquero e P. S. Almeida. "A Survey of Distributed Data Aggregation Algorithms". Em: *CoRR* abs/1110.0725 (2011). arXiv: 1110.0725. URL: http://arxiv.org/abs/1110.0725.

[0] A. Lakshman e P. Malik. "Cassandra: a decentralized structured storage system". Em: *ACM SIGOPS Operating Systems Review* 44.2 (2010), pp. 35–40.

[11] J. Leitão, P. Á. Costa, M. C. Gomes e N. M. Preguiça. "Towards Enabling Novel Edge-Enabled Applications". Em: *CoRR* abs/1805.06989 (2018). arXiv: 1805.06989. URL: http://arxiv.org/abs/1805.06989.

[12] J. Li, J. Stribling, T. Gil, R. Morris e M. Kaashoek. "Comparing the Performance of Distributed Hash Tables Under Churn". Em: mar. de 2004. DOI: 10.1007/978-3-540-30183-7_9.

[0] S. Madden, M. J. Franklin, J. Hellerstein e W. Hong. "{TAG}: {Tiny} {AGgregate} Queries in Ad-Hoc Sensor Networks". Em: *Proceedings of the{USENIX} Symposium on Operating Systems Design and Implementation* (2002), pp. 131–146. URL: xxx.

[13] P. Maymounkov e D. Mazières. "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric". Em: *Peer-to-Peer Systems*. Ed. por P. Druschel, F. Kaashoek e A. Rowstron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 53–65. ISBN: 978-3-540-45748-0.

[14] S. Newman. *Building microservices: designing fine-grained systems*. "O'Reilly Media, Inc.", 2015.

[0] R. V. A. N. Renesse, K. P. Birman e W. Vogels. “Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining”. Em: *ACM Transactions on Computer Systems* 21.2 (2003), pp. 164–206.

[15] A. Rowstron e P. Druschel. “Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems”. Em: *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer. 2001, pp. 329–350.

[16] A. Rowstron, A.-M. Kermarrec, M. Castro e P. Druschel. “Scribe: The Design of a Large-Scale Event Notification Infrastructure”. Em: *Networked Group Communication*. Ed. por J. Crowcroft e M. Hofmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 30–43. ISBN: 978-3-540-45546-2.

[17] *Self-driving Cars Will Create 2 Petabytes Of Data, What Are The Big Data Opportunities For The Car Industry?* URL: https://datafloq.com/read/self-driving-cars-create-2-petabytes-data-annually/172.

[18] W. Shi, J. Cao, Q. Zhang, Y. Li e L. Xu. “Edge Computing: Vision and Challenges”. Em: *IEEE Internet of Things Journal* 3 (out. de 2016), pp. 1–1. DOI: 10.1109/JIOT.2016.2579198.

[19] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek e H. Balakrishnan. “Chord: a scalable peer-to-peer lookup protocol for internet applications”. Em: *IEEE/ACM Transactions on Networking (TON)* 11.1 (2003), pp. 17–32.

[0] D. Stutzbach e R. Rejaie. “Understanding churn in peer-to-peer networks”. Em: *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. ACM. 2006, pp. 189–202.

[0] S. Tarkoma, C. E. Rothenberg e E. Lagerspetz. “Theory and Practice of Bloom Filters for Distributed Systems”. Em: *IEEE Communications Surveys Tutorials* 14.1 (2012), pp. 131–155. ISSN: 2373-745X. DOI: 10.1109/SURV.2011.031611.00024.

[20] S. Yi, Z. Hao, Z. Qin e Q. Li. “Fog computing: Platform and applications”. Em: *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*. IEEE. 2015, pp. 73–78.

[21] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph e J. Kubiatowicz. “Tapestry: A Resilient Global-Scale Overlay for Service Deployment”. Em: *IEEE Journal on Selected Areas in Communications* 22 (jul. de 2003). DOI: 10.1109/JSAC.2003.818784.

[22] Zhiyong Xu, Rui Min e Yiming Hu. “HIERAS: a DHT based hierarchical P2P routing algorithm”. Em: *2003 International Conference on Parallel Processing, 2003. Proceedings*. 2003, pp. 187–194. DOI: 10.1109/ICPP.2003.1240580.

[23] J. Zhu, D. Chan, M. Prabhu, P. Natarajan e H. Hu. “Improving Web Sites Performance Using Edge Servers in Fog Computing Architecture”. Em: mar. de 2013, pp. 320–323. ISBN: 978-1-4673-5659-6. DOI: 10.1109/SOSE.2013.73.

Apêndice A

Appendix 2 Lorem Ipsum

Anexo I

Annex 1 Lorem Ipsum