Nuno Morais

Master of Science

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

Índice

Capítulo 1

Introduction

1.1 Context

Nowadays, the Cloud Computing paradigm is the standard for development, deployment and management of services, most software present in our everyday life such as Google Apps, Amazon, Twitter, among many others is deployed on some form of cloud service. This paradigm has proven to have massive economic benefits that make it very likely to remain permanent in future of the computing landscape. Cloud Computing provides the illusion of unlimited computing power, which and has revolutionized the way developers, users and businesses rationalize about building and deploying applications [1].

However, 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. However, when computation resides 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.

The aforementioned aspects 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 which act as an "edge"along the path between the data source and the DC. It addresses the increasing need for supporting interaction between cloud computing systems and mobile or IoT applications [29], and allows the emergence of novel edge-enabled applications (e.g. traffic management, smart city management, mobile games, among others).

Additionally, systems that require real-time processing of data may be not be feasible with cloud computing, Google's self-driving car generates 1 Gigabyte every second [27], while a Boeing 787 will create around 5 gigabytes of data per second [10]. If this data were to be processed in real-time (e.g. towards self-driving), it would be infeasible to transport it to cloud and back.

1.2 Motivation

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. Contrary to the cloud, edge environments tend to be highly dynamic, devices have constrained computational power and their connections are often limited in capacity and reliability.

Developing an efficient general compute platform for edge environments is still an open challenge in Edge Computing. A crucial requirement towards this is performing **resource management**, which consists of keeping track of the tasks to perform and manage the utilization of computational resources of each device. Resource management systems are extensibly applied and optimized towards managing clusters (e.g. Mesos [14], Yarn [33]), however, those solutions are tailored towards small numbers of homogenous resource-heavy devices, which is the opposite of the edge environment.

Instead, a resource monitoring solution for these environments must be capable of federating very large numbers of devices, while leveraging heterogeneity to build a hierarchical infrastructure which combines naturally with the device taxonomy. An infrastructure capable of performing the aforementioned tasks successfully has strong applicability in todays world (e.g. Cloud providers, Smart Cities, among others).

A particularly hard task in resource management is **scheduling**, which consists in distributing tasks among nodes in the system, ideally, the task distribution must promote a balanced resource usage among nodes in the system. One of the popular solutions is transporting all the data towards a centralized point and redistribute the tasks among nodes. However, this presents a centralized point of failure and a point of contention in a large scale system.

Alternatively, nodes with only partial knowledge of the system (ideally without sending any additional messages) must be able to autonomously offload tasks towards neighbors. However, offloading tasks of varied complexity in heterogeneous nodes is not an easy task. And the accuracy of the knowledge each peer has dictates how efficiently they can offload them such that the system remains balanced in the face of environment changes.

Given this, we believe that peers must integrate a robust decentralized **resource monitoring system**. finish

1.3 Expected Contributions

The expected contributions, as will be further detailed in section 3, derive from the aforementioned challenges:

- Devise a decentralized monitoring infrastructure for edge devices which employs a topology tailored towards edge environments, and a combination of monitoring and aggregation techniques in a natural way which promotes load-balancing and networking locality.
- Evaluate the designed infrastructure through simulation (e.g. iFogSim or PeerSim) and compare the performance with similar systems.
- Design an experimental scenario to test the system under various scenarios. For this, we must design or adapt an existing scheduler and test the feasibility of the solution by deploying applications and collecting metrics about the overlay, the potential reduction in cost and the quality of service.

1.4 Document structure

This document is structured in the following manner:

Chapter 2 focuses on the related work, first section covers edge computing in further detail, next we cover P2P systems and the different types of topology management protocols. Third section studies the different types of resource location architectures, namely how to efficiently find a specific peer in the system and common techniques towards performing efficient searches over networks composed by a large number of devices. Fourth section covers aggregation techniques and popular implementations of these systems era para meter isto em monitoring?. Finally, last section covers popular resource monitoring systems.

Chapter 3 further explains the proposed contribution and proposes the work plan for the remainder of the thesis.

Capítulo 2

Related Work

In this chapter we provide further context about the Edge Computing paradigm and study related work to the identified challenges towards creating a unified infrastructure which performs monitoring and management of resources in the edge environment. First, we study the taxonomy of resources in the edge of the network, and describe how they can be employed (Section 2.1).

Then, after understanding the environment taxonomy, we describe common approaches towards integrating those devices in an efficient abstraction layer. It is paramount that such layer provides efficient communication patterns, traffic locality and load balancing (according to device heterogeneity). As such, we will cover the different techniques of topology management and present popular implementations (Section 2.2).

Following, we analyze how nodes can find resources (e.g. other peers, services or even computing power) in the aforementioned layer. We present popular implementations for each type of **resource location architecture** in the state of the art (Section 2.3).

Next, we study how to **monitor** the state of the devices and the tasks running on them, we study popular techniques of monitoring device and task status, and how to aggregate those values (Section 2.4).

Lastly, we study how the monitoring data can be used to distribute and offload tasks. We discuss the differences between performing resource management in the Edge instead of the Cloud. Lastly, we discuss related towards offloading tasks in Edge environments.

2.1 Offloading computation to the Edge

In this chapter we provide context about related work towards decentralizing computation from the Cloud. First, we define how we characterize Edge Computing and discuss how it is related to edge-related paradigms. Following, we study the taxonomy of the environment and focus on which computations each device can perform.

2.1.1 Edge Computing

As previously mentioned, edge computing calls for the processing of data in the edge of the network, specifically servicing IoT devices and performing computations on behalf of cloud services. This proposes to address battery life constraints, lower bandwidth consumption (which in turn lowers infrastructure costs) as well as address data privacy and safety [28].

Many approaches have already leveraged on some form of Edge computing in the past. **Cloudlets** [34] are an extension of the cloud computing paradigm beyond the DC, which consists of deploying resource rich computers near the vicinity of users that provide cloud functionality.

A limitation of cloudlets is that because they are specialized computers, they cannot not guarantee low-latency ubiquitous service provision, and consequently cannot satisfy QoS of large hotspots of users. Cloudlets have become a trending subject, and have been employed towards resource management, Big Data analytics, security, among others.

Content Distribution networks [23] (CDNs) emerged to address the overwhelming utilization of network bandwidth and server capacity that arose with bandwidth-intensive content (e.g. streaming HD video). In short, CDNs consist of specialized high bandwidth servers strategically located at the edge of the network, these servers replicate content from a certain origin and serve it at reduced latencies, effectively decentralizing the content delivery.

Additionally, many paradigms have emerged which propose to solve similar problems to the Edge Computing paradigm. **Fog Computing** [4] proposes to provide compute, storage and networking services between end devices and traditional cloud computing data centers, typically, but not exclusively located at the edge of the network. Fog computing is interchangeable with our vision of all devices acting as an "edge"contributing towards computing, however, with a bigger emphasis on providing infrastructure towards sensor networks that produce large amounts of data, and less in how to properly integrate the whole infrastructure.

Osmotic Computing [35] envisions the automatic deployment and management of inter-connected microservices on both edge and cloud infrastructures. Osmotic computing envisions edge devices employing an orchestration technique similar to the process of "osmosis". Translated, this consists in dynamically detecting and resolving resource contention via coordinated microservice deployments, furthermore, this paradigm is focused towards ensuring and maintaining quality of service.

Multi-access edge computing (MEC) formerly known as mobile-edge cloud computing, is a network architecture that proposes to provide fast-interactive responses for mobile applications. It solves this by employing the network edge (e.g. base stations and access points) to provide compute resources for latency-critical mobile applications [21].

MEC is a subset of our edge computing vision, although with a higher focus on communications technology and how to offload the computation from mobile to the cloud and not vice-versa.

2.1.1.1 Edge Environment Taxonomy

Similar to [18], we classify edge device according to 3 main attributes: **capacity** refers to computational, storage and connectivity capabilities of the device, **availability** consists in the probability of a device being reachable, and finally, **domain** characterizes the way in which a device may be employed towards applications in short, if the device can support the whole *applicational domain* or only the activities of a single user (*user domain*) .

Tabela 2.1: Taxonomy of the edge environment

Level	Category	Availability	Capacity
L0	Cloud Data Centers	High	High
L1	ISP, Edge & Private DCs	High	High
L2	5G Towers	High	Medium
L3	Networking devices	High	Low

Level	Category	Availability	Capacity
L4	Priv. Servers & Desktops	Medium	Medium
L5	Laptops	Low	Medium
L6	Mobile devices	Low	Low
L7	Actuators & Sensors	Varied	Low

Table 2.1 shows the categories of edge devices, we assign levels to categories as a function of the distance from the cloud infrastructure. Coincidently, the levels are correlated to the number of devices and their computational power, where higher levels tend to have more devices that are closer to the origin of the data and have lower computational power.

Levels 0 and 1 *cloud and edge DCs* offer pools of computational and storage resources, that can dynamically scale to support the operation of edge-enabled applications (e.g. serving as an optimization reference point for edge devices). Both of these have high availability and large amounts of storage and computational power, as such, there is no limitations on which type of computations these devices can perform.

Level 2 is composed of *5G cell towers*, which serve as access points for mobile devices, **Level 3** also consists of *networking devices*, although with lower capacity than those in level 2. Devices in both levels have high availability, and they can still contribute to the applicational domain, however in a limited fashion (e.g. coordinate resource management, host a microservice, or just act as a gateway for mobile devices).

Level 4 Consists of *private servers and desktops*, it is the first level where devices belong to the user domain. Devices in this level have medium capacity and availability, and can perform a varied amount of tasks on behalf of devices in higher levels (e.g. compute on behalf of smartphones, act as logical gateways or just cache data).

Level 5 consists of *laptops*, which are also on the user domain, and can perform a role similar to devices in level 4, although with lower availability and capacity. The main differentiating factor with devices in level 4 is that laptops are battery-powered, which means that energy consumption must also be taken into account whenever monitoring and computing on these devices.

Level 6 consists of *tablets and mobile devices*, devices in this level act as producers and consumers of data and belong to the user domain. Because because of their low capacity, availability and short battery life, mobile devices are limited in how they can perform computations and contribute towards edge applications. Aside from caching user data, common usages are filtering or aggregation of data generated from devices in level 7.

Finally, **level 7** consists of *actuators*, *sensors and things*, these devices are the most limited in their capacity, and varied availability. *Things* act both as data producers and consumers towards edge-enabled applications. They enable limited forms of computation in the form of aggregation and filtering.

2.1.2 Discussion

Intuitively, the lower the level the harder it is to employ devices towards applications. Devices in levels 6 and 7 are especially restricted due to having lower availability and computational power, however, these can still be used in specific scenarios (e.g. an application with very low computational overhead but real-time latency requirements).

Devices in levels 0-5 are potential candidates towards building the resource monitoring system we intend to create. Given the low availability of devices in higher levels, they are not very suitable towards overlay that supports the resource monitoring system, as they could not contribute much towards the system and would incur churn. This can be circumvented by employing devices in other levels as gateways for mobile devices and *things*.

2.2 Topology Management

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, mainly the taxonomy of overlay networks.

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).

Instead, 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**.

2.2.1 Evaluating topologies

Given that the accumulation of the neighboring relations among peers forms a graph, one can define a set of metrics to measure graph-related 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.

2.2.2 Taxonomy of overlay networks

There are two main approaches to build decentralized P2P systems, these are commonly categorized in two categories: structured and unstructured. Furthermore, within those categories we have identified the following sub-categories: flat and hierarchical.

2.2.3 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 their low maintenance cost, given that nodes can easily create neighboring relations and replace failed ones. Consequently, this is the type of overlay which offers better resilience to churn [31] (participants concurrently entering and leaving the system).

2.2.3.1 Flat unstructured Overlays

A flat unstructured overlay is an overlay where peers contribute evenly towards a common system goal. These overlays attempt to have even degree distributions while providing good connectivity. A prime example of a flat unstructured overlay which is highly resilient to churn and catastrophic failures is **Hyparview** [0]. Hyparview (Hybrid Partial View) gets its name from maintaining two exclusive views: the *active* and *passive* view. What distinguishes these views is their maintenance strategy.

The *passive view* is a larger view which consists of a random set of peers in the system, this view is maintained by a simple gossip protocol which periodically gossips a message to a random peer in the active view. This message contains a subset of the neighbors of the sending node and a time-to-live (TTL), the message is forwarded in the system until the TTL expires. In contrast, the *active view* is a smaller view (around log(n)) created during the bootstrap of the protocol, and actively maintained by monitoring peers with a TCP connection (effectively making the active view connections bidirectional and acting as a failure detector). Whenever peers from the active view fail, nodes attempt to replace them with nodes in the passive view.

Hyparview achieves high reliability even in the face of high percentage of high node failures (up to 80-90% of all nodes). This is highly desirable in edge environments. However, due to battery constraints, many devices cannot actively maintain TCP connections, which limits the applicability of Hyparview towards levels 0-4 of the taxonomy described in 2.1.1.1. Hyparview is often used as a *peer sampling service* for other protocols which rely on the connections from the active view to collaborate (e.g. PlumTree [0]).

2.2.3.2 Unstructured Hierarchical Overlays

Unstructured hierarchical are characterized as overlays where peers have different tasks in the system, this is an easy way to accommodate device heterogeneity while potentially increasing the performance of the system.

The canonical example is **super-peers**, which are peers that have increased capacity and stability, that are commonly assigned towards disseminating queries throughout the system or caching file locations.

Super-peers have proven their effectiveness in reducing the number of peers that have to exchange messages, which by consequence raises system scalability. This is the approach taken by Gia [0] to improve the scalability of Gnutella [0]. However, this approach is inefficient when finding rare resources in the system.

SOSP-Net (Self-Organizing Super-Peer Network) [0] proposes to optimize super-peer networks by organizing super-peers in a topology that reflects the semantic similarity of peers with similar interests. Super-peers maintain cache references to files which were recently requested by weak peers, while the weak peers maintain caches containing super-peers that satisfied most of its requests. This yields good cache hit ratios, and load balancing. However, the original paper does not specify how super-peers communicate among themselves, nor the election process of super-peers.

Overnesia [0]

2.2.4 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.

2.2.4.1 Flat structured overlays

2.2.4.2 Hierarchical structured overlays

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.

2.2.5 Discussion

Unstructured overlays are an attractive option to federate edge devices, as they can easily adapt to dynamic environments, this facilitates employing resource-constrained devices towards the system.

2.3 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.

2.3.1 Query taxonomy

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.

2.3.2 Query dissemination

A basic requirement towards forming these overlays is performing efficient information exchange among nodes, as previously mentioned, it would be undesirable if all nodes communicate with all other nodes in the system.

To circumvent this, throughout the years many communication patterns arose which are specifically tailored towards different requirements.

First, **Flooding**

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.

Random Walks are a dissemination strategy that attempts to minimize the communication overhead that accompanies flooding. Instead of flooding, a random walk consists of a message with a TTL that is randomly forwarded one peer at a time throughout the network. Walks may also take a biased paths in the system based on information accumulated by peers, through aggregation, this is called a **random guided walk**. Random guided walks forward queries to neighbors that are more likely to have answers [7]. Another approach to bias walks is to use bloom filters [32], which are space-efficient probabilistic data structures that support set membership queries. There are many other techniques of performing guided walks, often tailored for application system needs.

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

2.3.3 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 [6] 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.

2.3.4 Decentralized Architectures

2.3.4.1 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, among others) 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 [17] and Amazon's Dynamo [8] 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 [30] 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 [19], 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 [3] 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 [37] 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 [11] 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 [25] 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 [26] and PAST [9]. 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 [22] 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 [13] 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 [36] 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).

2.3.4.2 Unstructured Overlays

2.3.4.3 Hybrid approaches

Curiata & **Build One Get One Free**

2.3.5 Discussion

2.4 Resource monitoring

The goal of this section is further identify the challenges that arise from creating a monitoring infrastructure tailored for general computing in edge environments. For this, we start by identifying the functional requirements of monitoring systems within edge computing scenarios.

2.4.1 Monitoring requirements

Scalability, a scalable monitoring system can handle a remarkable number of monitored devices and services. This property is paramount towards edge computing environments because of the necessity of managing a wide variety of parameters which need to be monitored across a largely decentralized environment. Existing centralized monitoring tools fail to distribute the monitoring load, which in turn leads to a single point of failure.

Non-intrusiveness, following the edge computing viewpoint of performing simple lightweight tasks, the monitoring solution should follow this approach. Given this, a monitoring tool which adopts minimal processing, consumes low memory and generates low amounts of traffic is essential.

Interoperability, edge computing aims at a cooperative deployment of applications interconnected over both cloud and edge infrastructures. Ideally, companies should be able to deploy components across a wide range of infrastructures owned by different providers.

Robustness, the monitoring system must be robust in the face of failures or network partitions. Especially in the edge environment, where devices are scattered and restricted in capacity.

Live Migration Support, with the current offers of virtualization technologies, there is a wide variety in resource management options, such as VMs or containers. For edge-enabled applications, live migration is a highly desirable feature, as it is a tool towards providing highly available applications.

2.4.2 Device monitoring

To adapt edge computing applications to changes in the environment and ensure application QoS requirements, it is necessary to tailor the monitoring system to support the whole spectrum of underlying infrastructures (section 2.1)

2.4.3 Service monitoring

2.4.4 Aggregation

Aggregation is an essential building block towards monitoring 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.

Towards monitoring edge devices and tasks, aggregation is paramount, examples of usages are (e.g. computing the average latency of the closest available service that meets a certain criteria; counting nearby available computing resources that can be used to offload services, or identify hotspots by aggregating the average system load in certain areas).

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

Tabela 2.2: popular aggregation functions in function of decomposability and duplicate sensitiveness

2.4.5 Properties of aggregation functions

Given this, it is important to understand the taxonomy aggregation functions. There are two properties of aggregation functions: *decomposable functions* and *duplicate sensitive* 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 2.2 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).

2.4.6 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.

2.4.6.1 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.

2.4.7 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

2.4.8 Discussion

2.5 Resource management

2.5.1 Schedulers

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.

[3] 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.

[4] F. Bonomi, R. Milito, J. Zhu e S. Addepalli. "Fog computing and its role in the internet of things". Em: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. 2012, pp. 13–16.

[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.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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] P. Garbacki, D. H. Epema e M. Van Steen. "Optimizing peer relationships in a super-peer network". Em: *27th International Conference on Distributed Computing Systems (ICDCS'07)*. IEEE. 2007, pp. 31–31.

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

[13] 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.

[14] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker e I. Stoica. "Mesos: A platform for fine-grained resource sharing in the data center." Em: *NSDI*. Vol. 11. 2011. 2011, pp. 22–22.

[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.

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

[0] J. Leitao, J. Pereira e L. Rodrigues. "HyParView: A Membership Protocol for Reliable Gossip-Based Broadcast". Em: *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. 2007, pp. 419–429. DOI: 10.1109/DSN.2007.56.

[0] J. Leitao, J. Pereira e L. Rodrigues. "Epidemic broadcast trees". Em: *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*. IEEE. 2007, pp. 301–310.

[18] J. Leitão, P. Á. Costa, M. C. Gomes e N. Preguiça. "Towards Enabling Novel Edge-Enabled Applications". Em: 732505 (2018). arXiv: 1805.06989. URL: http://arxiv.org/abs/1805.06989.

[0] J. C. A. Leitão e L. E. T. Rodrigues. "Overnesia: a resilient overlay network for virtual super-peers". Em: *2014 IEEE 33rd International Symposium on Reliable Distributed Systems*. IEEE. 2014, pp. 281–290.

[19] 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.

[21] Y. Mao, C. You, J. Zhang, K. Huang e K. Letaief. "A Survey on Mobile Edge Computing: The Communication Perspective". Em: *IEEE Communications Surveys & Tutorials* PP (ago. de 2017), pp. 1–1. DOI: 10.1109/COMST.2017.2745201.

[22] 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.

[23] G. Peng. "CDN: Content distribution network". Em: *arXiv preprint cs/0411069* (2004).

[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.

[25] 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.

[26] 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.

[27] *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.

[28] W. Shi, J. Cao, Q. Zhang, Y. Li e L. Xu. "Edge Computing: Vision and Challenges". Em: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646. ISSN: 2372-2541. DOI: 10.1109/JIOT.2016.2579198.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O’Malley, S. Radia, B. Reed e E. Baldeschwieler. “Apache Hadoop YARN: yet another resource negotiator”. Em: *SOCC ’13*. 2013.

[34] T. Verbelen, P. Simoens, F. De Turck e B. Dhoedt. “Cloudlets: Bringing the Cloud to the Mobile User”. Em: *Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services*. MCS ’12. Low Wood Bay, Lake District, UK: Association for Computing Machinery, 2012, 29–36. ISBN: 9781450313193. DOI: 10.1145/2307849.2307858. URL: https://doi.org/10.1145/2307849.2307858.

[35] M. Villari, M. Fazio, S. Dustdar, O. Rana e R. Ranjan. “Osmotic computing: A new paradigm for edge/cloud integration”. Em: *IEEE Cloud Computing* 3.6 (2016), pp. 76–83.

[36] 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.

[37] 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.