Human's Cloud

A community cloud served by a P2P overlay network on top of the web platform

David Dias, david.dias@computer.org

Lisbon Tech, University of Lisbon

Abstract. Grid computing has been around since the 90's, it's fundamental basis is to use idle resources in geographically distributed systems in order to maximize it's efficiency, giving researchers access to computational resources to perform their studies. This approach quickly grew into non grid environments, causing the appearance of projects such as SETI@Home or Folding@Home, that use volunteered shared resources and not only institutionalized data centers as before, creating the concept of Public Computing. Today, after having volunteering computing as a proven concept, we face the challenge of how to create a simple way for people to participate in this community efforts and even more importantly, how to reduce the friction of adoption by the developers and researchers to use this resources for their applications. This work explores current ways of making an interopable way of end user machines to communicate, using new Web technologies, creating a simple API that's familiar to those used to develop applications for the Cloud, but with resources provided by a community and not by a company or institution.

Keywords: Cloud Computing, Peer-to-peer, Voluntary Computing, Cycle Sharing, Decentralized Distributed Systems, Web Platform, Javascript, Fault Tolerance, Reputation Mechanism,

1 Introduction

1.1 Cloud Computing

1.2 Peer-to-Peer

"An application is peer-to-peer if it aggregates resources at the networks edge, and those resources can be anything. It can be content, it can be cycles, it can be storage space, it can be human presence.", C.Shirky [31]

1.3 Web platform

2 Objectives

3 Related Work

The purpose of this section is to show the state of the art of the research topic, namely: Volunteer Computing, Cloud Computing, P2P Networks and the Web Platform

3.1 Cloud computing and Open Source Cloud Platforms

3.2 Volunteered resource sharing

3.2.1 Hybrid and Community Clouds

3.2.2 Cycle and Storage Sharing, using Volunteer Computing Systems

3.2.3 Peer-to-Peer Networks Architectures - Efficient resource discovery mechanisms are fundamental for a distributed system success, such as grid computing, cycle sharing or web application infrastructures[25], although in the centralized model, by keeping data bounded inside a data center, we have a stable and scalable way for resource discovery, this does not happen in a P2P network, where peers churn rate can vary greatly, there is no way to start new machines on demand for high periods of activity, the machines present are heterogeneous and so is their Internet connectivity, creating an unstable and unreliable environment. To overcome this challenges, several researches have been made in order to optimize how data is organized across all the nodes, improving the performance, stability and the availability of resources. The following paragraphs will describe the current state of the art P2P organizations, typically categorized in P2P literature as Unstructured or Structured[23], illustrated in Figure 1.

***Unstructured* -** We call 'Unstructured' to a P2P system that doesn't require or define any constraint for the placement of data, these include Napster, Kazaa and Gnutella, famous for it's file sharing capabilities, where nodes can share their local files directly, without storing the file in any specific Node. There is however a 'caveat' in the Unstructured networks, by not having an inherent way of indexing the data present in the network, performing a lookup results of the cost of asking several nodes the whereabouts of a specific file or chunk of the file, creating a huge performance impact with an increasing number of nodes. In order to overcome this, Unstructured P2P networks offer several degrees of decentralization, one example is the evolution from Gnutella 0.4[9] to Gnutella 0.6 [33][28], which added the concept of super nodes, entities responsible for storing the lookup tables for the files in parts of the network they are responsible for, increasing the performance, but adding centralized, single points of

failure. [25] classifies Unstructured networks into two types: deterministic and non-deterministic, defining that in a deterministic system, we can calculate before hand the number of hops needed to perform a lookup, knowing the predefined bounds, this includes systems such as Napster and BitTorrent[6], in which the file transfers are decentralized, the object lookup remains centralized, keeping the data for the lookup tables stored in one place, which can be gathered by one of two ways : (i) peers inform directly the index server the files they have; or (ii) the index server performs a crawling in the network, just like a common web search engine, this gives this network a complexity of O(1) to perform a search, however systems like Gnutella 0.6, which added the super node concept, remain non deterministic because it's required to execute a query flood across all the super nodes to perform the search.

Structured with Distributed Hash Tables - Structured P2P networks have an implicit way of allocating nodes for files and replicas storage, without the need of having any specie of centralized system for indexing, this is done by taking the properties of a cryptographic hash function [2][18][24], such as SHA-1[7], which applies a transformation to any set of data with a uniform distribution of possibilities, creating an index with O(log(n)) peers, where the hash of the file represents the key and gives a reference to the position of the file in the network. DHT's such as Chord[32], Pastry[29] and Tapestry[37], use a similar strategy, mapping the nodes present in the network inside an hash ring, where each node becomes responsible for a segment of the hash ring, leveraging the responsibility to forward messages across the ring to his 'fingers'(nodes that it knows the

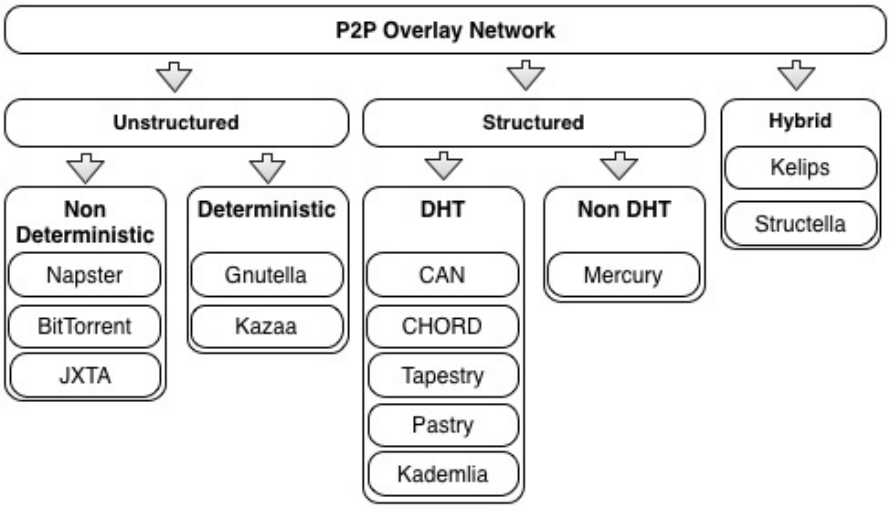

Fig. 1. Different types of P2P Overlay networks organizations

whereabouts). Kademlia[21] organizes it's nodes in a balanced binary tree, using XOR as a metric to perform the searches, while CAN[16] introduced and a several dimension indexing system, in which a new node joining the network, will split the space with another node that has the most to leverage. Evaluating the DHT Structured P2P networks raises identifiable issues/challenges, that result as the trade-off of not having an centralized infrastructure, responsible for railing new nodes or storing the meta-data, these are: (i) generation of unique node-ids is not easy achievable, we need always to verify that the node-id generated doesn't exist, in order to avoid collisions; (ii) the routing table is partitioned across the nodes, increasing the lookup time as it scales. Table 1, showcases a comparison of the studied DHT algorithms.

P2P system	**Overlay Structure**	**Lookup Protocol**	**Networking parameter**	**Routing table size**	**Routing complexity**	**Join/leave overhead**
Chord	1 dimension, Hash ring	Matching key and NodeID	n= number of nodes in the network	O(log(n))	O(log(n))	$O(\log(n)^2)$
Pastry	Plaxton style mesh structure	Matching key and prefix in NodeID	n= number of nodes in the network, b=base of identifier	$O(\log_b(n))$	$O(b \log_b(n)+b)$	O(log(n))
CAN	d-dimensional ID Space	Key value pair map to a point P in the D-dimensional space	n= number of nodes in the network, d=number of dimensions	O(2d)	$O(d\ n^{1/2})$	O(2d)
Tapestry	Plaxton style mesh structure	Matching suffix in NodeID	n=number of nodes in the network, b=base of the identifier	$O(\log_b(n))$	$O(b \log_b(n)+b)$	O(log(n))
Kademlia	Binary tree	XOR metric	n=number of nodes, m=number of different bits (prefix)	O(log(n))	$O(\log_2(n))$	not stable

Table 1. Summary of complexity of structured P2P systems

Structured without Non-Distributed Hash Tables - Mercury[4], a structured P2P network that uses a non DHT model, was design to enable range queries over several attributes that data can be dimensioned on, which is desired on searches over keywords in several documents of text. Mercury design

offers an explicit load balancing without the use of cryptographic hash functions, organizing the data in a circular way, named 'attribute hubs'.

Hybrid - NOTE: Not sure if should include this, doesn't really include anything that new

3.2.4 Fault Tolerance, Load Balancing, Assurance and Trust Volunteer resource sharing means that we no longer have our computational infrastructure in a confined and well monitored place, this introducing new challenges that we have to address [19] to maintain the system running with the minimum service quality, this issues can be: scalability, fault tolerance, persistence, availability and security[36] of the data and that the system doesn't get compromised. This part of the document serves to describe the techniques implemented in previous non centralized systems to address this issues.

Fault Tolerance, Persistence and Availability are one of the key challenges in P2P community networks, due to it's churn uncertainty, making the system unable to assume the availability of Node storing a certain group of files. Previous P2P systems offer a Fault Tolerance and Persistence by creating file replicas, across several Nodes in the network, one example is PAST[12][30], a system that uses PASTRY routing algorithm, to determine which nodes are responsible to store a certain file, creating several different hashes which corresponds to different Nodes, guaranteeing an even distribution of files across all the nodes in the network. DynamoDB[8], a database created by Amazon to provided an scalable NOSQL solution, uses a storage algorithm, inspired by the CHORD routing algorithm, in which stores file replicas in the consequent Nodes, in order to guarantee easy lookup if one of the Nodes goes down. The strategy presented by the Authors of PAST to provide high availability, is an intelligent Node system, that use a probabilistic model, able to verify if there is an high request for a file, deciding to keep a copy and avoiding to overload the standard Node with every request that is made.

Load Balancing in an optimal state, can be defined as having each node sharing roughly 1/N of the total load inside the network, if a Node has a significantly hight load compared with the optimal distribution, we call it a 'heavy' node. There has been some research to find a optimal way to balance the load inside a P2P network, namely:

- Power of Two Choices[5] - Uses multiple hash functions to calculate different locations for an object, opts to store it in the least loaded node, where the other Nodes store a pointer. This approach is very simple, however it adds a lot of overhead when inserting data, however there is a proposed alternative of not using the pointers, which has the trade-off of increasing the message overhead at search.

- Virtual Servers[26] - Presents the concept of virtualizing the Node entity to easy transfer it amongst the machines present in the P2P network. It uses two approaches, 'one-to-one', where nodes contact other Nodes inside the network with the expectation of being able to trade some of the load, shifting a virtual server, or an 'one-to-many/many-to-many' in which a directory of load per node is built, so that a node can make a query in order to find it's perfect match to distribute his load. Virtual Servers approach has the major issue of adding a extra amount of work to maintain the finger tables in each node.
- Thermal-Dissipation-based Approach[27] - Inspired by the heat expansion process, this algorithm shifts nodes position inside the hash ring windows of load responsibility, in a way that the load will implicitly flow from a node to it's close peers.
- Simple Address-Space and Item Balancing[17] - It's an iteration over the virtual servers, by assigning several virtual nodes to each physical node, where only one of which is active at a time and this is only changed if having a different nodeId distribution in the network brings a more load balanced hash ring

S. Rieche, H. Niedermayer, S. Gtz and K. Wehrle from the University of Tbingen, made a study comparing this different approaches in a scenario using the CHORD routing algorithm, using a SHA-1 as the hashing function, with 4096 nodes and 100.000 to 1.000.000 documents and executing up to 25 runs per test, the results can be observed in the Figure 2

Assurance and Trust in a P2P network is an interesting challenge due to the lack of control over the machines that are willing to share with their resources, in order to achieve it, several strategies have been developed to maintain the integrity of the data using Cryptography, Reputation modeling schemes based on it's node previous record and also economic models, that resemble our own economy, but to share and trade computational resources.

Starting with the Cryptographic techniques, storage systems such as PAST give the option to the user to store encrypted content, disabling any other user, that does not have the encryption key, to have access to the content itself, this is a technique that comes from the Client-Server model, adapted to P2P environment, however, other cryptography technique benefits such as user authorization and identity, cannot be directly replicated into a P2P network without having a centralized authority to issue this validations, one of the alternatives is using distributed signature strategy, known as Threshold Cryptography [10], where an access is granted if validated if several peers (a threshold), validates it's access, one implementation of Threshold Cryptography can be see in a P2P social network[1] in order to guarantee privacy over the contents inside the network.

Trust in a P2P system, as mentioned, is fundamental to it's well behaved functioning, not only in terms of data privacy, but also in giving the deserved resources to the executions that mostly need them, avoiding misbehaved peer intentions that can be a result of an Attack to jeopardize the network, one

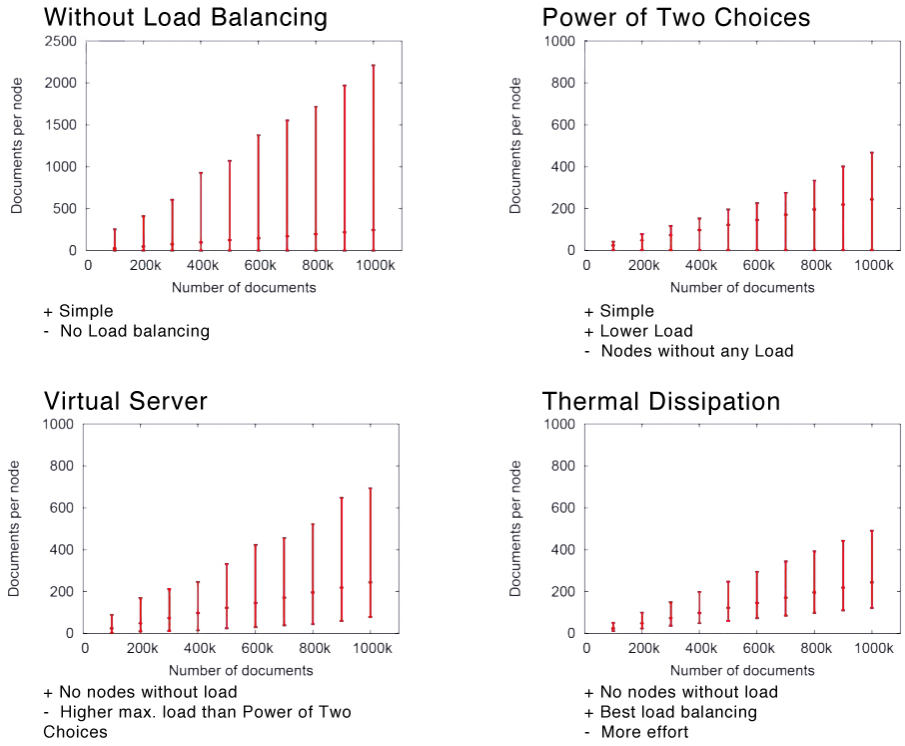

Fig. 2. Load balancing approaches comparison

example is the known Sybil attack[11]. To achieve a fair trust sharing system, several metrics for a reputation mechanism have been developed [20], these can be seen in Table 2.

Reputation Systems		
Information Gathering	Scoring and Ranking	Response
Identity Scheme	Good vs. Bad Behavior	Incentives
Info. Sources	Quantity vs. Quality	Punishment
Info. Aggregation	Time-dependence	
Stranger Policy	Selection Threshold	
	Peer Selection	

Table 2. Reputation system components and metric

Incentives for sharing resources[15] can in the form of money rewards, greater speed access(used in Napster and some bittorrent networks) or it can be converted to a interchangeable rate to trade for more access to resources, giving the birth of economic models[14][35], that model the traded resources as a currency in which a peer has to trade in order to use the network.

3.3 Resource sharing using the Web as platform

3.3.1 Web Platform

3.3.2 Previous attempts on cycle sharing through web platform The first research of browser-based distributed cycle sharing was performed by Juan-J. Merelo, Juan Lupion and Fernando Tricas, which introduced a Distributed Computation on Ruby on Rails framework[22]. The system used a client-server architecture in which clients, using a browser would connect to a endpoint, where they would download the jobs to be executed and sent back the results. In order to increase the performance of this system, a new system[13] of browser-based distributed cycle sharing was creating using Node.js as a backend for very intensive Input/Output operations[34], with the goal of increased efficiency, this new system uses normal webpages(blogs, news sites, social networks) to host the client code that will connect with the backend in order to retrieve and execute the jobs, while the user is using the webpage, this concept is known as parasitic computing[3], where the user gets to contribute with his resources without having to know exactly how, however since it's Javascript code running on the client, any user has access to what is being processed and evaluate if it presents any risk to the machine.

4 Architecture

In this section it is described what is expected to be implemented. First, we present an overall of the architecture of Human's Cloud(Figure 3), followed by

a description of the individual components, responsible for separate tasks such as: developer front end (API), storage system, job scheduling techniques, the architecture of the system at the node level and finally, the proposed reputation system to assign different responsibilities for each Node.

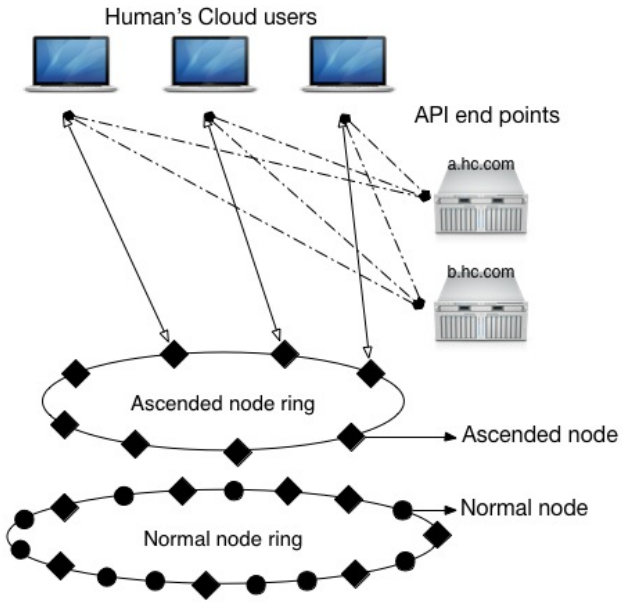

Fig. 3. Human's cloud overall architecture

Human's Cloud proposed architecture has the goal of enabling the user to develop on top of computing and storage resources provided by volunteers, without having to change their current application conventions.

Nodes (volunteer computers), are divided into two PASTRY DHTs with the purpose of separating the nodes with storage responsibility from the ones with only computing responsibility. The reason behind this decision is due to the high churn rate in a P2P network, keeping the files in Nodes have proven to be more trustworthy for staying longer in the network makes the more robust by keeping the file replica level stable, this also reduces the message overhead that would require to keep the replica level in a more inconsistent environment. Never the less, the more volatile nodes are perfect for short computing operations, till they proven to be trustworthy to 'ascend' in the network.

4.1 Client API

The client API offers the traditional and expected functions for a Cloud Computing services in a very Unix like way, which developers are familiar with, these are:

- **$ hcls** - List files in a directory
- **$ hccd** - Traverse in storage directories
- **$ hcget** - Get an object stored
- **$ hcput** - Store an object
- **$ hcjob** - Initialize a job

In order to execute this commands, a 'connect' action must be issued first in which the user will request to one of the API endpoints for one point of contact in the Ascended nodes ring, that will act as mediator for all the user communications. Human's Cloud provides several API end points as a fault tolerance mechanism, inspired by DNS, so the user has always a way to discover one node to contact.

To avoid node overload, the API endpoints have enabled a load balancing system that will assign different nodes as mediators to different users, based in the amount of activity the node was subjected too, the more activity, the less probability it will be selected next.

The creation of a job is described by a script, presenting the objects that will be manipulated and the several assets that will be used as steps in order to process the job, such as:

"hcjob /path/to/files — step1 [— step2] — /path/to/output"

for example:

[GRAB ONE MANTA EXAMPLE TO HERE]

The assets, can be one of the present in Human' Cloud or one provided by the user, respecting the service policies of not using functionalities that may cause some misbehavior of the node executing the job.

4.2 Storage

Human's Cloud storage happens in what it's named, the "Ascended node ring", this nodes have an higher reliability, making the storage system more stable, without the need of constantly burning computer cycles to maintain the files replica level.

Data stored in nodes can be:

- File metadata (name of the file, size, location of the chunks, chunks hash).
- File chunks
- Directories metadata - This way, hcls can be more efficient
- Job information (state, issuer, workflow)
- Reputation log

We classify storage nodes into two types, the 'sKeeper', responsible for holding the metadata of the file and hashing each chunk to identify the 'sHolder', nodes responsible to store the chunk into their system. This approach mitigates the possibility of having an high unbalanced storage distribution, diving each file in equal chunks across several nodes. As we can see in Figure 4, each chunk gets hashed more than one time with a different hash function, the purpose is to identify several Nodes that will be responsible to store a replica, also, in order to increase the fault tolerance of the system, we replicate the 'sKeeper' responsibility in the 2 following nodes in the hashring, so if one of these fails, another is assigned.

Fig. 4. A file partitioned in several chunks, each with its corresponding hashes that correspond to nodeIds

In Figure 5, we can find the 'sKeeper' and 'sHolder' relationship. Only the sKeeper performs the chunk hashing and stores the information in the file lookup table, this happens one single time for each chunk, reducing several computer cycles for the consequent searches.

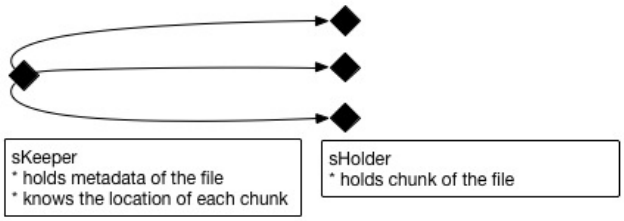

Fig. 5. Representation of the Node responsible for the file(sKeeper) and it's individual chunk holders(sHolders)

Each store file is chunked as soon as it enters the network, this mitigates the risk that would be present if we were transferring files with considerable sizes all at once, starving the network and the node's heap. The only point where the file gets glue together again is when it leaves the network and sent to the user.

Human's Cloud adapts the Load Balancing virtual server's method, by using the same strategy of global load, but by transferring files between sHolders and not an entire virtual server, updating the respective sKeeper accordingly. The reasons behind

Files are storage as objects in a indexedDB type storage, provided by the leveljs module.

4.3 Job Scheduling

4.4 Node Level

Figure 6

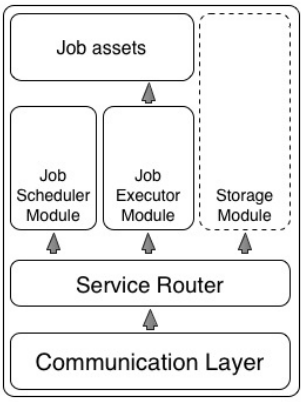

Fig. 6. Human's Cloud Node

4.5 Reputation Mechanism

The reputation mechanism present will enable the network to identify the nodes that show more availability and have the necessary means to ascend and take a more important role. In order to evaluate each node, we define several metrics, these are: uptime, number of job completions, network throughput and computational resources available, being the uptime, the most important, to assure stability.

The reputation of each node is stored with its node identifier on the 'ascended hash ring', each time a job is completely successfully, his score gets updated and in case it reaches the required level to ascend, the jKeeper that was updating his score will enable and deploy the remaining features(storage and job schedule module) he needed to join the ascended group.

5 Evaluation

5.1 Lorem ipsum

Yada yada yei

6 Conclusions

6.1 Lorem ipsum

Excepteur sint

References

1. Youssef Afify. Access Control in a Peer-to-peer Social Network Youssef Afify.
2. S Bakhtiari and J Pieprzyk. Cryptographic Hash Functions : A Survey 1 Introduction. pages 1–26.
3. a L Barabási, V W Freeh, H Jeong, and J B Brockman. Parasitic computing. *Nature*, 412(6850):894–7, August 2001.
4. Ashwin R Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury : Supporting Scalable Multi-Attribute Range Queries. pages 353–366.
5. John Byers, Jeffrey Considine, and Michael Mitzenmacher. Simple Load Balancing for Distributed Hash Tables. pages 80–88.
6. Bram Cohen. The BitTorrent Protocol Specification, 2009.
7. Cisco D. Eastlake, 3rd Motorola; P. Jones Systems. RFC 3174 US Secure Hash Algorithm 1 (SHA1), 2001.
8. Giuseppe Decandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo : Amazons Highly Available Key-value Store. pages 205–220, 2007.
9. Protocol Definition. The Gnutella Protocol Specification v0 . 4. *Solutions*, pages 1–8, 2003.
10. Y Desmedt and Y Frankel. Threshold cryptosystems. *Advances in Cryptology-CRYPTO'89 ...*, 1990.
11. John R Douceur. The Sybil Attack. pages 1–6.
12. Peter Druschel and Antony Rowstron. PAST A large-scale , persistent peer-to-peer storage utility. pages 75–80, 2001.
13. Jerzy Duda and W Dubacz. Distributed evolutionary computing system based on web browsers with javascript. *Applied Parallel and Scientific Computing*, 2013.
14. Pedro Filipe and Goldschmidt Oliveira. Gridlet Economics : Modelo e Políticas de Gestão de Recursos num Sistema para Partilha de Ciclos Gridlet Economics : Resource Management Models and Policies for Cycle-Sharing Systems Pedro Filipe Goldschmidt Oliveira Dissertação para a obtenção do Grau de. 2011.
15. Philippe Golle, Kevin Leyton-brown, Ilya Mironov, and Mark Lillibridge. Incentives for Sharing in Peer-to-Peer Networks. pages 75–87, 2001.
16. Mark Handley and Richard Karp. A Scalable Content-Addressable Network.
17. David R. Karger and Matthias Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures - SPAA '04*, page 36, 2004.
18. David Kargerl, Tom Leightonl, and Daniel Lewinl. Consistent Hashing and Random Trees : Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web *. pages 654–663.
19. Georgia Koloniari and Evaggelia Pitoura. Peer-to-Peer Management of XML Data : Issues and Research Challenges. 34(2):6–17, 2005.
20. Sergio Marti and Hector Garcia-molina. Taxonomy of Trust : Categorizing P2P Reputation Systems. (April 2005):1–20.

21. Petar Maymounkov and David Mazières. Kademlia: A Peer-to-peer Information System Based on the XOR Metric.
22. Juan-j Merelo, Antonio Mora-garcía, Juan Lupión, and Fernando Tricas. Browser-based Distributed Evolutionary Computation : Performance and Scaling Behavior Categories and Subject Descriptors. pages 2851–2858, 2007.
23. Dejan S Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, Zhichen Xu, and J I M Pruyne. Peer-to-Peer Computing. Technical report, 2003.
24. Bart Preneel. The State of Cryptographic Hash Functions. pages 158–182, 1999.
25. Rajiv Ranjan, Aaron Harwood, and Rajkumar Buyya. A study on peer-to-peer based discovery of grid resource information. *. . . , Australia, Technical Report GRIDS . . .* , pages 1–36, 2006.
26. Ananth Rao, Karthik Lakshminarayanan, Sonesh Surana, and Richard Karp. Load Balancing in Structured P2P Systems. 0225660:68–79, 2003.
27. S. Rieche, L. Petrak, and K. Wehrle. A thermal-dissipation-based approach for balancing data load in distributed hash tables. *29th Annual IEEE International Conference on Local Computer Networks*, pages 15–23.
28. M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. *Proceedings First International Conference on Peer-to-Peer Computing*, pages 99–100, 2002.
29. Antony Rowstron and Peter Druschel. Pastry : Scalable , Decentralized Object Location , and Routing for Large-Scale Peer-to-Peer Systems. pages 329–350, 2001.
30. Antony Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. *Proceedings of the eighteenth ACM symposium on Operating systems principles - SOSP '01*, page 188, 2001.
31. C. Shirky. Clay shirkys writings about the internet. In http://www.shirky.com/.
32. Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan Ý. Chord : A Scalable Peer-to-peer Lookup Service for Internet. pages 149–160, 2001.
33. R. Manfredi T. Klingberg. RFC - Gnutella 0.6 Protocol Specification, 2002.
34. Stefan Tilkov and Steve Vinoski Verivue. Node.js : Using JavaScript to Build High-Performance Network Programs. 2010.
35. Vivek Vishnumurthy, Sangeeth Chandrakumar, and G Emin. KARMA : A Secure Economic Framework for Peer-to-Peer Resource Sharing.
36. Dan S Wallach. A Survey of Peer-to-Peer Security Issues.
37. Ben Y Zhao, John Kubiatowicz, and Anthony D Joseph. Tapestry : An Infrastructure for Fault-tolerant Wide-area Location and Routing. (April), 2001.