Service allocation methodologies for contributory computing environments

Guillem Cabrera Añon

gcabreraa@uoc.edu

PhD Thesis Dissertation

Doctoral Programme in Network and Information Technologies

Advisors: Dr. Joan Manuel Marquès Puig
Dr. Ángel Alejandro Juan Pérez

DPCS Research Group

Internet Interdisciplinary Institute

Universitat Oberta de Catalunya

Barcelona, July 2014

Abstract

Cloud computing technologies fostered the growth and great success of many Internet applications. The huge computational power and storage capacity offered by their world-wide distributed ensemble of digital warehouses allowed the appearance of novel services offered directly to consumers, being online social networks one of the most bestselling among them. At the same time, commodity domestic computers benefited from new hardware advances and access network increased their offered bandwidth. Nevertheless, the common end-user usage does not harness all the available capabilities from these new computers and networks. Several initiatives emerged trying to seize the surplus or idle computer and network capacities from end-users. Peer-to-peer systems became extremely popular in file-sharing applications and volunteer computing schemes helped the scientific community by performing calculations within several specific projects.

The contributory computing paradigm was born aiming to utilize the contributed computing capacities. The main idea was that users would donate their resources to a self-managed community that would support general purpose computing applications, like the ones held by current clouds. Previous research proved the model feasibility and defined the main components needed to support decentralized and self-managed contributory communities.

This thesis is focused on the improvement of contributory platforms. We reviewed the status of the contributory computing model within the distributed computing. From this review, we identified the need for better node selection mechanisms for application replica hosting regarding different parameters. First, we discussed service availability as a key factor for long-lived service survival. Consequently, we proposed a methodology to properly select hosts in order to offer a given availability level. Second, we noticed an important research trend in minimizing the environmental footprint of computational infrastructures called *green computing*. We studied the energy consumption of contributory platforms and presented two different mechanisms to minimize these consumptions: a scheduling algorithm to reduce the global makespan of ordered-task executions and an extension of our availability-aware allocation method that includes also energy consumptions in the host selection stage. Finally, we looked at the effect of the underlying network connecting the participant nodes in a community. Contributory systems share their self-provision philosophy with the novel *Community Networks*. In these networks, there exist some facilities that publicly offer information about network topology and host status. These openness provides data that is usually not available in commercial access networks and allowed new studies in the field. We presented a methodology to select those hosts that are closer to as many potential customers in a network, in such a way the network distance is minimized.

All these contributions aim at offering new tools to enhance the contributory computing model and attract new services and users.

Resum

Les tecnologies de computació al núvol (*cloud computing*) han esdevingut un model de gran èxit per a moltes aplicacions d'Internet. L'enorme capacitat computacional que ofereix la multitud de centres de processament distribuïts a tot el món van propiciar l'aparició de nous serveis per a usuaris finals. Les xarxes socials són l'exemple més clar d'aquest nou conjunt de serveis. Al mateix temps, els ordinadors domèstics s'han beneficiat també dels nous avenços en maquinari i les connexions a Internet a través de les xarxes d'accés comercials han augmentat considerablement el seu ample de banda. Tot i així, l'ús que la majoria d'usuaris fa d'aquests ordinadors i aquestes connexions està lluny d'aprofitar tota la potència i ample de banda disponibles. Han aparegut moltes iniciatives per aprofitar aquesta capacitat computacional o ample de banda sobrant o ociós. Les xarxes de igual a igual (*peer-to-peer*) en són el cas d'ús amb més èxit a l'hora de compartir fitxers, però els sistemes de computació voluntària (*volunteer computing*) també han ajudat a la comunitat científica a assolir fites en projectes concrets.

El model de computació contributiva (*contributory computing*) va néixer amb la intenció d'utilitzar les capacitats computacionals donades pels usuaris finals. La idea principal era que els usuaris donarien els seus recursos a una comunitat autogestionada que oferiria aplicacions de propòsit general, com les suportades per les plataformes de computació al núvol ja existents. La recerca prèvia en aquest sentit ja ha demostrat la viabilitat d'aquest model i ha definit els components bàsics per a suportar comunitats contributives distribuïdes i autogestionades.

Aquesta tesi se centra en la millora del funcionament d'aquestes comunitats contributives. En primer lloc s'estudia l'estat del model de computació contributiva dins del marc dels sistemes distribuïts. D'aquest estudi, s'identifica la necessitat de mètodes per a la selecció de nodes per a hostatjar o suportar determinades aplicacions en funció de diferents paràmetres. Després es debat la importància de la disponibilitat dels serveis com a element clau per a la supervivència de serveis de llarga vida. En conseqüència, es proposa una metodologia per a seleccionar els nodes adequats per a garantir un cert nivell de disponibilitat de servei. A continuació s'assenyala una tendència important en el camp de la recerca de la computació sostenible (*green computing*) com a aquella que intenta minimitzar l'impacte ambiental de les infraestructures computacionals existents. S'estudia el consum energètic de les plataformes contributives i es presenten dos mecanismes per a minimitzar aquests consums: un algoritme de planificació per a reduir els temps de computació globals en l'execució de tasques ordenades i una extensió de la metodologia referent a la disponibilitat de serveis per a incloure també el consum energètic com a factor de decisió en la selecció de nodes en sistemes contributius. Finalment s'estudia l'efecte de la xarxa que connecta els nodes en el funcionament dels serveis contributius. Les comunitats contributives comparteixen la filosofia d'auto-aprovisionament i autogestió amb el nou model de xarxes comunitàries (*community networks*). Aquestes xarxes disposen d'una mínima infraestructura de monitorització que permet oferir de manera lliure informació sobre la topologia de xarxa i l'estat de nodes i enllaços. Aquesta informació no està disponible de manera habitual en les xarxes d'accés comercial i permet començar noves línies de recerca en aquest sentit. Aquesta tesi presenta una metodologia per a seleccionar aquells nodes que quedin el més propers possible a tots els clients potencials d'un servei a la xarxa, de manera que la distància de xarxa en número de salts es minimitza.

Totes aquestes contribucions pretenen oferir noves eines que millorin el model de computació contributiva per tal d'atraure nous serveis i usuaris.

Acknowledgments

En primer lloc vull agrair als meus directors la seva ajuda durant tot el procés que ha culminat en aquesta tesi doctoral. A en Joan Manuel Marquès, per confiar en mi a l'iniciar aquest viatge, per guiar-me a través de tot el procés i per ajudar-me a arribar a bon port amb la seva direcció. A n'Àngel Juan per la seva determinació i implicació en tots els projectes de recerca que comença i per la seva inacabable energia. Sense la vostra ajuda, res del que he fet al llarg de tots els anys d'aquest doctorat hagués estat possible.

També vull donar les gràcies a tots els companys amb els que he treballat durant aquests anys: el primer de tots, en Daniel Lázaro, pels seus consells als inicis i per la primera empenta en el món de la recerca que em va saber donar; después a Alejandra Pérez Bonilla, por estar siempre dispuesta a aclarar conceptos y aconsejarme como seguir adelante en todo momento; a Hebert Pérez Rosés, por su inagotable paciencia y su predisposición innata para enseñar aquello que me era desconocido; y a Enoc Martínez, por su ayuda en el desarrollo e implementación de algunos de los algoritmos de este trabajo. No em vull oblidar d'en Sergio González, per la seva gran voluntat i iniciativa en tot allò que hem fet junts; ni d'en Davide Vega, ex-company de classe i excel·lent col·laborador en tot allò en que hem coincidit durant el doctorat, tot i treballar en grups de recerca diferents.

Moltíssimes gràcies a en David Sanchez, na Laia Descamps i en Marc Esteve per les bones estones que hem compartit aquests anys. Estaré sempre en deute amb na Debora Lanzeni, na Nadia Hakim, en José Cáceres i en Biel Company per les llargues estones de companyia en la soledat d'un hospital. I want to thank as well my office-mates, Enosha Hettiarachchi, Amna Qureshi and Pere Tuset for keeping a great work-environment during these years at IN3.

I am very grateful to Scott Grasman for welcoming me at RIT and hosting me there for few months. His research advises and his determined support made my stay at Rochester a great and profitable experience. I would also thank all the members of the INFORMS@RIT society for all their interesting and funny initiatives I had the pleasure to join.

Finalment, gràcies a la meva família, en especial als meus pares i al meu germà. El vostre recolzament ha estat fonamental per a acabar aquesta tesi doctoral.

Contents

List of Figures

List of Tables

List of Algorithms

List of Publications

Articles in Journals

[P1] D. Lázaro, J.M. Marquès, G. Cabrera, H. Rifà-Pous, and A. Montané. Hornet: Microblogging for a contributory social network. *IEEE Internet Computing*, 16(3):37-45, May-June 2012. **Indexed in ISI SCI**, 2012 IF 2.039, Q1 (Computer Science, Software Engineering).

[P2] G. Cabrera, H. Pérez-Rosés, A. Juan, and J.M. Marquès. Operations research in green internet computing: state of the art and open challenges. *Journal of Applied Operational Research, Special Issue: OR Applications in Natural Resources Management*, 5(4), December 2013.

[P3] G. Cabrera, A. Juan, D. Lázaro, J.M. Marquès, and I. Proskurnia. A simulation-optimization approach to deploy internet services in large-scale systems with user-provided resources. *Simulation: Transactions of the Society for Modeling and Simulation International*, 90(6):644-659, March 2014. **Indexed in ISI SCI**, 2012 IF 0.692, Q3 (Computer Science, Software Engineering), Q4 (Computer Science, Interdisciplinary Applications).

Articles in Conference proceedings

[P4] G. Cabrera, H. Pérez-Rosés, A. Juan, and J.M. Marquès. Sustainable internet services in contributory communities. In *Advances in Artificial Intelligence*, *Proceedings of the 2013 Multiconference of the Spanish Association for Artificial Intelligence*, pages 260-268. Springer Lecture Notes in Artificial Intelligence, Madrid (Spain), September 2013. **Indexed in ISI Web of Science and Scopus**, 2012 SJR 0.332, Q2.

[P5] G. Cabrera, H. Pérez-Rosés, A. Juan, and J.M. Marquès. Promoting Green Internet computing throughout simulation-optimization scheduling algorithms. In *Proceedings* *of the Winter Simulation Conference 2013*, Washington DC (USA), December 2013. **Indexed in ISI Web of Science and Scopus**, 2011 SJR 0.372, Q2.

[P6] G. Cabrera, S. González-Martín, A. Juan, S. Grasman, and J.M. Marquès. Combining biased random sampling with metaheuristics for the facility location problem in distributed computer systems. In *Proceedings of the Winter Simulation Conference 2014*, Savannah (USA), December 2014. Accepted for publication.

Presentations in conferences and workshops

[P7] A. Juan, G. Cabrera, J.M. Marquès, and C. Aragonés. Despliegue eficiente de servicios fiables sobre sistemas distribuidos. In *Proceedings of the XIX Jornadas de Concurrencia y Sistemas Distribuidos*, La Granja de San Ildefonso (Segovia, Spain), June 2011.

[P8] G. Cabrera and K. Sagar. Availability issues in large-scale distributed computer systems. In *2011 CYTED-HAROSA Workshop and Meeting on Applied Optimization and Distributed Computing*, Barcelona (Spain), July 2011.

[P9] G. Cabrera. Hornet: Microblogging for a contributory social network. In *2012 IN3-HAROSA Workshop for Young Researchers*, Barcelona (Spain), July 2012.

[P10] G. Cabrera. Maximizing availability by eficiently deploying Internet services over non-dedicated resources. In *2012 CYTED-HAROSA International Workshop on Simulation-Optimization and Internet Computing*, Valparaíso (Chile), November 2012.

[P11] G. Cabrera, H. Pérez-Rosés, A. Juan, and J.M. Marquès. Reduciendo el consumo energético en comunidades contributivas. In *Proceedings of the XXI Jornadas de Concurrencia y Sistemas Distribuidos*, San Sebastián (Spain), June 2013.

[P12] G. Cabrera. Internet service allocation in contributory environments. In *2013 ICSO-HAROSA International Workshop on Simulation-Optimization and Internet Computing*, Barcelona (Spain), July 2013.

[P13] G. Cabrera, S. González-Martín, A. Juan, and J.M. Marquès. Despliegue de replicas de servicios cercanas al usuario en entornos comunitarios. In *Proceedings of the XXII*

Jornadas de Concurrencia y Sistemas Distribuidos, Morella (Castelló, Spain), June 2014.

[P14] G. Cabrera. A multi-start based algorithm with iterated local search for the uncapacitated facility location problem. In *20th Conference of the International Federation of Operational Research Societies*, Barcelona (Spain), July 2014. Accepted for publication.

1 Introduction

1.1 Motivation

Nowadays, desktop computers are commodity goods affordable for most citizens and organizations in developed countries. This is the main reason behind the high growth of computing as a common tool in every-day's life for people around the world. As well, the spread of domestic and mobile network connections with increasing bandwidth popularized the use of Internet to an extent that nowadays one can even talk about commodity networks.

Besides the gained popularity in the domestic environments, the reduced prices for powerful computers fostered the appearance of large computational infrastructures composed by vasts amounts of them. These aggregations, often known as clusters or server farms, propitiated the intensive usage of computational power for many different tasks or operations in enterprises and scientific projects. These aggregations of computers are often centralized and composed mostly by homogeneous and dedicated resources, well-maintained and provisioned under strict maintenance policies. The exploitation of these infrastructures requires some strategic decisions, such as hardware interconnection schemes, data or service allocation statements and usage policies. These technologies were the basis for the Grid [1] and Cloud [2] computing paradigms, on which several distant cluster instances can be combined to create larger and world-wide distributed systems.

The spread of commodity networks and their increasing capabilities also modified the usage patterns of these connections. Novel applications, such as Peer-to-Peer systems [3] or Volunteer computing platforms [4] changed the client-server paradigm to a more distributed scenario, on which data and services are no longer placed in strategic regions of the network. In these scenarios, computing resources are no longer homogeneous and no maintenance policies can be assured, since the computers are owned and controlled by end users. Moreover, they are geographically dispersed across wide-area networks and the number of involved nodes can be much higher than in clusters. This large scale, together with the heterogeneity, the dynamism, the stochastic behavior and the dispersion of the involved resources makes the operation of these systems to be even more challenging than in the case of distributed clusters.

From a different perspective, different processes involved in the operation of distributed systems require a selection of some of the involved nodes to perform certain actions. This selection can be regarded as a scheduling, location or allocation problem, depending on the nature of the task. Thus, this decision process can be modeled as a classical optimization problem, on which the choice of different elements results in different quality systems and an objective is satisfied. The objective could be regarded for example as minimizing the processing time for a task scheduler, maximizing scavenged CPU cycles, minimize network distances among replicas, guaranteeing a certain level of availability or minimizing the energy consumption.

In a homogeneous and controlled environment, this choice is arbitrary and has no great impact on the performance and efficiency of the system. However, when the involved nodes have different capabilities and are placed in distant locations, a proper selection among them could result in a more efficient or a less resource-demanding system. Furthermore, dealing with very large scale, dynamic and stochastic systems also complicates the problem and disables many of the classical analytic approaches for optimization. Heuristic or metaheuristic procedures showed to be a valuable tool to cope with similar problems in different research topics, such as logistics, transportation or supply chain management, but also in the distributed computing field [5, 6, 7].

This thesis explores some of the described problems in the novel paradigm of contributory computing, in which non-dedicated computing resources are aggregated to support open-access network services in a way similar to current cloud computing platforms. Specifically, this dissertation studied how host availability affects the survivability of multi-component services and how it can be maximized; how the energy consumed by distributed platforms could be minimized; and how the network position of a given replica influences on the bandwidth usage. We propose several heuristic methods to address the aforementioned problems.

Figure 1.1 shows a topic map with the studied fields and their relation with the chapters in this dissertation.

1.2 The computer distributed systems context

The distributed computing paradigm can be generally defined as the aggregation of multiple autonomous computing resources interacting through a network connection to achieve a common goal [8]. This definition is general enough to include systems of very different nature. The following sections describe those ones that preceded and inspired the contributory computing model.

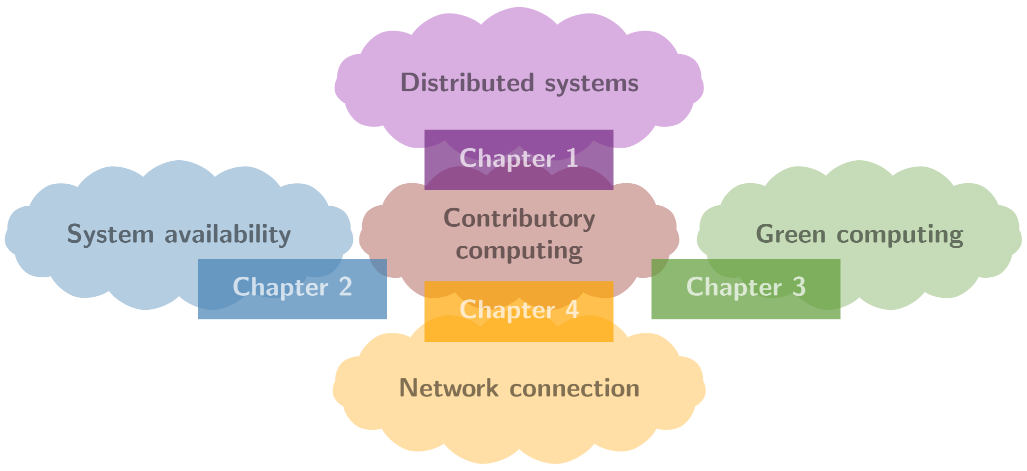

Figure 1.1: Dissertation topics

1.2.1 Grid Computing

Since the beginning of the 1990s decade, the ongoing scientific and research needs surpassed the capacity of current single workstations. Hence, scientists allover the globe pursued the goal of aggregating dispersed computational resources to perform large computational tasks, aiming at making computer power as easy to access as the electrical power grid. It was from this terminology that the metaphor of *grid computing* became a keyword to identify accessible computer distributed systems [1, 9].

The most important challenge achieved by the earliest grid computing research was the interoperability of distributed systems [10]. A significant number of different and independent technologies or protocols were used by each computing infrastructure. This was the largest barrier the computer scientists had to overcome in order to allow users and organizations to access resources from other institutions.

A whole new set of protocols and tools were designed and implemented by the community, leading to the *de facto* standard Globus Toolkit[1]. This tool defined the open standards and protocols to manage different remote processes, covering all the needed functions and procedures (resource management, rights and security, task execution, file transfers or storage).

Thanks to the standardization efforts, computing resources belonging to many different institutions organized in different administrative domains could be federated into Virtual

[1]`http://toolkit.globus.org/`

Organizations and easily accessed by users involved in other organizations. Thus, large and computational extensive tasks were processed by gathering heterogeneous, geographically dispersed and loosely coupled resources.

While no limitation on the supported tasks was explicitly described, in practice its usage was restricted to the execution of large batch tasks in the scientific or research fields and caused no viable grid computing providers emerged in the open market [11].

1.2.2 Cloud Computing

One step further on computer distributed systems was the appearance of the *cloud computing* paradigm during the first decade of the XXI century [2]. Its objective was the same as the original of the grid computing model: allow users to easily access unlimited computing resources.

The grid computing model focused on creating the standards to create Virtual Organizations that federated resources from different organizations to make them available to its members under certain usage policies. The cloud computing model aims at offering the same access to an apparently infinite amount of computing resources, but describes in contrast a pure provider-client model, in which users do not belong to any given organization and pay for the strict resources they utilize [12, 13]. This *pay-as-you-go* consumption model is the reason behind the cloud providers success in the last years and that it became an increasingly popular tool to outsource large computing infrastructures.

Cloud providers own and manage a vast amount of well-managed, well-provisioned, mostly homogeneous and dedicated computing resources, organized in large datacenters placed in strategic locations around the world. This distributed infrastructure is known as a *cloud*. Users can access to all these resources, combining several locations and different hardware features. No up-front investment is required and no over-provisioning is needed to face the peak load, as clients can pay exactly for the amount of resources their applications require at any time. Thus, capital expenses are not longer an entry barrier for Small and Medium Enterprises (SMEs) and scaling up becomes an easy, cheap and quick process for startups.

Different service models appeared as the providers developed their commercial offers. Mainly, they differ in the features offered by the platform itself and which applications should support themselves. Three of them can be clearly distinguished:

Software as a service also referred as ‘on demand software’, cloud providers maintain an application software and its needed infrastructure, in such a way the client accesses the software directly, with no knowledge or implication in the underlying layers. An example of this model is an online Enterprise Resource Planner (ERP) or an online office software suite.

Platform as a service cloud providers offer a full computing platform, operating and managing the underlying infrastructure. Typical platforms are operating systems, execution environments and database or webs servers.

Infrastructure as a Service cloud operators provide computers (either physical or virtual machines) on demand. Cloud users must manage their own operating systems and applications.

Besides the advantages of the clouds, they also present some important drawbacks that limit their expansion to even more applications [14]. Moving enterprise or private data to an external provider poses new security risks. Relying exclusively on a single external provider may become a restriction in terms of interoperability with other cloud solutions or products and could lock in companies to their providers, causing operative and economical issues to their business.

The exponential growth of cloud computing was the foundation for a plethora of new Internet services, some of them being just the re-implementation of already-existing local services exported to the cloud. The appearance of infinite resources enabled long-lived services composed by several components ready to interact and react to user actions. The clearest examples of these new generation of services are social networks (like Facebook[2] or Twitter[3]), massive online storage services (like Dropbox[4] or Google Drive[5]), online video platforms (like

[2]`http://www.facebook.com/`
[3]`http://www.twitter.com/`
[4]`http://www.dropbox.com/`
[5]`http://drive.google.com/`

YouTube[6] or Netflix[7]) or cloud-service providers (like Amazon Web Services[8] or Windows Azure[9] among others).

1.2.3 Peer-to-Peer Systems

A different evolution branch of computer distributed systems were the peer-to-peer (P2P) networks. In the traditional client-server model, one of the computers is responsible of providing a service (known as server) and the others (the clients) contact him to consume this service. On the contrary, in the P2P model all the components in the system are connected to each other as equal *peers* and establish connections with other components in the network. These connections enable peer interactions, on which both sides can act as client and server at the same time to request, obtain and exchange information.

The main features of these systems and what make them interesting for the research community and pose a challenge to predict system dynamics are the following [15]:

a) The *high degree of decentralization* they support, due to all the burden tasks in the system are performed by participating nodes, releasing any centralized node.

b) The *self-organization* in the system management, since no manual configuration should be need in the participating nodes.

c) The *fault tolerance* and *attack resilience*, thanks to the few or any critical nodes in the system and the large amount of hosts to target simultaneously in order to affect the system.

d) The *high scalability*, allowing the system to grow almost arbitrarily without manager intervention.

e) The *abundance of resources* contributed by the participating nodes and *diversity* in terms of hardware and software.

[6]`http://www.youtube.com/`
[7]`https://www.netflix.com/`
[8]`https://aws.amazon.com/`
[9]`http://www.windowsazure.com/`

f) The presence of *multiple administrative domains*, since most of the nodes would be owned and managed by individuals who freely join and leave the system. This trait causes indirectly a *geographical dispersion* on the resource locations.

One of the key challenges in any P2P system is to build an overlay with routing capabilities that operates properly under node churn. This overlay can be thought as a directed graph on which the nodes aware of each other have an edge between them and therefore can communicate directly though the network and can be structured or unstructured. In *(partly) centralized architectures*, nodes join the overlay by contacting a system controller and create a star topology around her. Later, additional connections can be introduced between collaborating hosts. In *decentralized systems*, the nodes join the system by contacting any of the already participant hosts, whose contact information is obtained by means of an external channel.

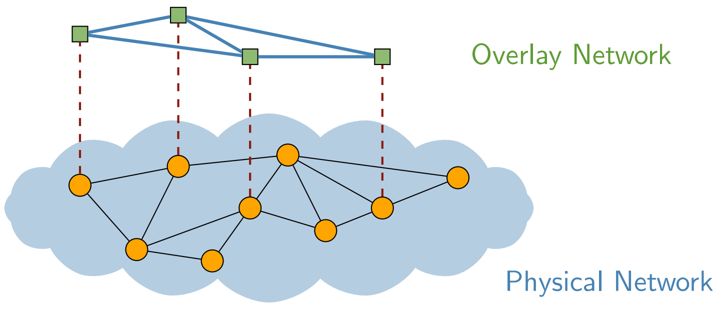

Figure 1.2: Peer-to-Peer overlay schema

System overlays are often classified on their structure. *Unstructured overlays* are those with no constraints on the links between hosts. Thus, nodes can connect to any of the other participants maintaining its connection degree in a given range and discovering new neighbors by random walks on the overlay. Accordingly, the network presents no particular structure.

In *structured overlays*, each host has a unique identifier (usually known as *key*) that determines its unique position in the overlay, which would present a specific structure. Keys are also used to determine node responsibilities, in such a way the whole key space is divided among all system participants. These techniques are commonly known as Key-based routing (KBR) and were the basis for the emergence of Distributed Hash Tables (DHT) in several flavors [16, 17, 18]. DHTs operate as a traditional hash table, inserting and distributing key/value pairs the participant nodes in the structured overlay.

P2P systems became popular at the end of the 1990s decade thanks to the well-known file sharing applications (like Naptser, Gnutella[10], eMule or BitTorrent[11] [19]), which make them appear as a source of copyright violations. However, different P2P approaches proved to be a valuable tool for large amount data distribution in corporate environments, with BitTorrent being used for content placement to its servers by Twitter [20] or Facebook [21]. Besides the file sharing applications, they support as well very disparate applications such as Internet video-telephony (Skype[12]), video streaming (CoolStreaming [22] or Sopcast[13]) or digital currencies (BitCoin [23]).

1.2.4 Volunteer Computing and Desktop Grids

Combining the system scale and the resource geographic dispersion of P2P networks and the aggregated computing power of the grid paradigm, the models of *volunteer computing* [4, 24] and *enterprise desktop grids* appeared, despite some authors include them in the P2P paradigm [15].

Volunteer computing and desktop grids are similar in the sense they aggregate surplus or idle computational resources from non-dedicated computers, often underutilized personal computers connected to the Internet and controlled by their individual owners. Nevertheless, in the former end-users voluntarily share their computational resources to be utilized by a third entity in a given project (often scientific projects such as SETI@home[14] or Folding@home[15]). The latter relies exclusively on resources inside of an organization network to make computations. In both cases, the architecture requires a set of centralized entities to coordinate the task execution and collect the obtained results.

The goal of volunteer computing is similar to that of the original grid computing, that is aggregating dispersed computational resources to perform large sets of computational tasks split in smaller independent jobs. The difference is that instead of exploiting only dedicated resources as in the grid, volunteer computing can combine dedicated and non-dedicated re-

[10] `http://rfc-gnutella.sourceforge.net/`
[11] `http://www.bittorrent.com/`
[12] `http://www.skype.com`
[13] `http://www.sopcast.org/`
[14] `http://setiathome.berkeley.edu/`
[15] `http://folding.stanford.edu/`

sources contributed by individual users to collect even more computational power[16]. The most successful implementations of this paradigm are BOINC [4] (used in the aforementioned projects), Condor [25] and Entropia [26].

Typically, volunteer computing systems are meant for a specific project. The project manager provides a central controller entity that manages the experimentation and distribute the computing tasks among the participants. To some extent, this is the same structure as the one in partly centralized P2P systems, on which a controller maintains the set of participating nodes and controls the system, but the burden tasks are performed in a distributed fashion by peers.

Project contributors can choose different sharing policies, based on their preferences. Typical policies would include sharing the whole capabilities when idle, a dynamic amount proportional to the load or a given fixed portion. They can also decide to freely change their contribution or withdraw the resource at any time. They run a software in their computers that notifies the project controller when their resources are available. Then, the project controller assigns a given independent task to be performed by this computer. As soon as the task is accomplished, the results are sent back. In case the computer becomes unavailable (either because the user switched it off, she started using it again or there was a network disconnection) and the task is not properly finalized, the same task can be re-assigned to another host.

1.3 The contributory computing paradigm

The *contributory computing* concept was defined by Lázaro as 'a novel paradigm for aggregating computational resources from different owners to be used collectively' [27]. In the framework of this proposal, he described as well the *contributory communities* as 'an aggregation of contributed resources which can be used as a platform to deploy computational services'. In this context, a service is regarded as any long-lived application executed on a set of computers that receives messages (queries), process them and sends messages back (responses), a powerful abstraction reinforced by the appearance of the service-oriented architectures [28, 29].

[16]as much as 8.12 petaFLOPS in BOINC in March 2014

This paradigm can be understood as the union of long-lived distributed services (similarly to Internet services on top of cloud infrastructures) with user-contributed resources (as the ones in volunteer computing or P2P networks) hosting them. We show the junction of the grid, cloud and volunteer computing models in Figure 1.3, being the contributory computing the intersection of cloud and volunteer computing.

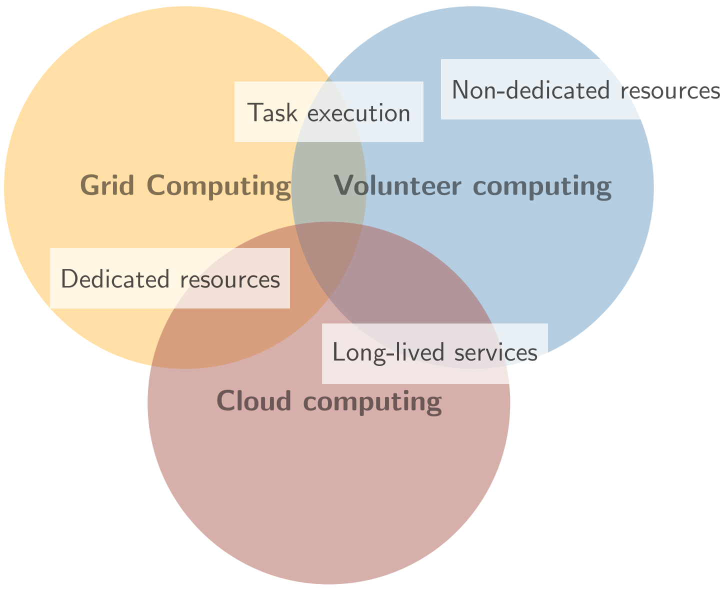

Figure 1.3: Basic scheme of cloud, grid, and volunteer computing feature intersections

As in the volunteer computing implementations, community members can contribute with different computing resources (such as commodity desktop computers or laptops, high-end servers or even mobile handhelds) and should choose their contribution policies. Gathering resources from domestic users implies also a geographical dispersion and the lack of maintenance or administration policies, as opposed to the managed server concentration in the cloud or grid environments.

All these traits make the contributory platforms to be extremely dynamic systems, with hosts joining and leaving without previous notice; the different hardware and software capabilities of each node increasing the heterogeneity of each community; and the network dispersion to increase the communication delays among hosts. Despite this dynamism and heterogeneity, contributory platforms were meant to be self-sufficient, relying only on user-contributed components. This can be achieved with a high degree of decentralization, replication strategies and mechanisms able to tolerate host churn, similarly as in the P2P systems.

The contributory computing model was designed with the idea of collective utilization of the resources. Users should contribute their resources unselfishly, with the only purpose of supporting the community. The same way, services should be available to all participants, regardless the amount of their contribution. In such an environment, it might seem reasonable to limit the resources available to a given user according its contribution as an incentive action. Despite such mechanisms or credit system would be profitable, the collective usage of the resources and its benefits for all users should prevent community members to use the resources in a selfish way.

1.3.1 Related proposals

Recent research trends presented different concepts similar in the philosophy of promoting non-dedicated resources utilization to build Information Technologies (IT) infrastructures or support Internet-like services.

Contributory Communities and CoDeS

Following his description of the contributory computing model, Lázaro designed and implemented CoDeS[17] [27] as a prototype middleware for service deployment in contributory communities. Besides describing all the desired features for fully-distributed and self-provisioned communities, he developed autonomous service deployment [30] and decentralized resource discovery [31] mechanisms for these systems. Also, he proposed an availability-aware resource selection method for stateless services [32] based on historical knowledge of host availability and replication strategies.

CoDeS was meant to aggregate sets of non-dedicated, Internet-distributed and heterogeneous computers to form a platform where general-purpose computing applications could be deployed. It provides tools to automatically manage resource aggregation and service deployment supported only by these resources.

It guarantees that management responsibilities and execution of services are automatically distributed among the nodes in a self-sufficient manner by exploiting FreePastry[18] [16]. Replica management responsibilities are assigned to hosts in the DHT by service indexes, thus

[17]`http://dpcs.uoc.edu/joomla/index.php/software/codes`

[18]`http://www.freepastry.org/`

maximizing decentralization and scalability. These nodes are known as *Service Managers* and are responsible to determine the most suitable nodes in the system to host a service, remotely deploy the service on them, monitor their status and create new replicas on node churn or system failure.

In his experimentation, Lázaro proved the feasibility of contributory communities in different circumstances with the CoDeS implementation deployed in PlanetLab [33]. He evaluated its resource discovery and service deployment mechanisms on different size communities under very different churn levels and concluded the contributory computing model is a feasible alternative to centralized and dedicated clouds. He also studied service-availability and proposed a replica allocation mechanism to ensure long-lived stateless service availability.

Being part of the original proposal of contributory computing, contributory communities supported by CoDeS fulfills most of the desired features of such a platform.

Community Cloud Computing

Marinos and Briscoe [34, 35] faced some of the limitations of the current cloud computing technologies proposing the novel concept of *Community Clouds*. They state the three key limitations of current clouds to be the following:

- Due to the concentration of services in large datacenters, major failures in these infrastructures cause the failure of a significant amount of them and generates long service downtimes.
- Being the infrastructure, the service and its data transferred to a cloud provider, the owner looses the control over them. This forces the clients to accept the conditions of the provider if they want to keep their services running.
- Environmental footprint of large digital warehouses cannot be neglected, because of their large energy consumption, their heat concentration and the waste generation of hardware renewals.

Their proposal tries to globally solve social, economic and environmental aspects of the cloud paradigm by replacing the proprietary clouds with contributed-resource communities organized in their architectural model.

Despite some of these ideas are aligned with the contributory computing model, this initiative fails on going further than a written proposal and providing an actual implementation.

Nebulas

Nebulas were defined as aggregations of voluntary resources donated by end-users to create a dispersed and less managed cloud [36]. Their goal is not to replace current clouds, but rather support them in particular situations where they fail.

First, the authors state current clouds are largely centralized and are inefficient when dealing with dispersed-data-intensive applications (those working with data spread at multiple distant locations). Moving data to centralized digital warehouses is expensive in terms of cost, time and bandwidth. Second, clouds are often separated from user context and location. Even though Content Delivery Networks (CDN) can alleviate this issue, sensitive applications would perform much better if were closer to the network edges than in centralized locations in the center of the network. And third, cost matters are pointed as an entry barrier for developers and applications.

Authors claim their system would be suitable as a service experimental platform to deploy dispersed-data-intensive applications close to the place where the data is stored and would allow end-user to publicly share their own services. Thus, edge machines could support commercial clouds in such scenarios.

The Nebula concept was firstly experimented in [37] with an Internet weblog-scanning application. Then, in [38] the authors proposed a Map-Reduce [39] data-intensive experimentation in a partly centralized architecture using Native Client sandboxing [40] on top of PlanetLab nodes and a centralized and dedicated controller. The usage of this centralized entity allows them to introduce a locality-aware task scheduler when assigning reduce tasks to collaborating hosts. This scheduler uses information from a monitoring system of inter-node link bandwidths to reduce data transmission times. They execute the Map-Reduce framework to count word appearance in a large set of ebooks, which amounts 500 MBytes of data. With this experiment, they show their performance in a data intensive scenario outperforms two volunteer computing systems (BOINC [4] and BOINC-MapReduce [41]) executing the same experiment.

Non-dedicated resources harvesting, their collective usage and the general application support make Nebulas to be close to the contributory computing model. The need for a centralized entity makes them similar to volunteer computing, an idea reinforced by the fact the authors compare their proposal to two existing volunteer computing approaches.

Cloud@Home

Cunsolo et al. combined the cloud and volunteer computing paradigms, generalizing volunteer computing environments using cloud techniques and principles [42, 43]. Their vision is that user-contributed resources can form a cloud ready to inter-operate with current commercial ones, still maintaining a centralized structure inside each of the clouds.

Cloud@Home focuses on maintaining interoperability with commercial clouds, in such a way the Cloud@Home could buy and sell them resources according its the demand and its surplus capacity. They pose the need of Quality-of-Service (QoS) policies and Service Level Agreements (SLA) to regulate these cloud interactions.

Even though they propose the gathering of user-donated resources, their focus on the interaction with commercial clouds make them aside from the pure contributory computing model. Furthermore, neither implementation nor experimentation to prove their statements is provided by the authors.

Community Networks

Following the same philosophy of self-provisioned platforms, Community Networks (CN) [44] showed up as a novel model of open and self-managed communication infrastructures. This type of networks are built, operated and owned by their users themselves, without any strict global planning policy. Any user willing to join the network can decide which existing node to connect. Even though no centralized decisions rule their development, some initiatives work to sustain and improve the networks. These initiatives provide some basic facilities (such as monitoring infrastructure or some equipment in the network core) and perform some strategic contributions to guarantee the system survival, improve its performance or extend their features.

Most of the successful systems following these principles took advantage of the advances in wireless transmission technologies to create the so-called Wireless Community Networks (WCN). Nevertheless, wired technologies like fiber optics can and are also employed in some regions of the networks. The largest and most successful examples of these infrastructures are Guifi.net[19], Athens Wireless Metropolitan Network[20] and FunkFeuer[21].

In addition to the network itself, some of the current initiatives offer basic open-access services to their members. The most used one and the most important attraction for many users to join these networks is the Internet proxy access, but other services like IP-telephony, distributed storage, network monitoring or web servers might be also available. The Internet access is provided by contributors that donate their bandwidth to other users in the network by offering a proxy service and other services are often supported by enthusiasts that donate them to the community with no reward.

These services are currently deployed independently and managed manually by their contributors. If certain capabilities related to resource discovery, replica management or directory were offered in the community network, new services could thrive and therefore attract more users and resources to the network.

1.3.2 Targeted services

The definition describing the services aimed to be supported by contributory computing is open enough to include a variety of different applications. Still, being hosted by non-dedicated resources imposes restrictions not present in the grid or cloud environments.

The simplest example of such a service is an instance of a web-server hosting static pages. A single replica of this service is stateless, in the sense that any response to a request is independent from the others and that any equal request would be answered with the same response. Thus, services are easy to replicate and replace, which make them suitable to be hosted by non-dedicated hosts. Besides web-servers, typical examples would be a video transcoder or a language translator. This interpretation of a service allows also the execution of batch tasks as services in a community.

Stateful services can also be achieved by splitting them into three different layers (the presentation, the service logic and the data itself), each of these layers being deployed independently. An external persistent storage mechanism can be seized to permanently save data

[19]`http://guifi.net/`
[20]`http://www.awmn.net/`
[21]`http://www.funkfeuer.at/`

and status information and the presentation layer can be handled by the client resources. Hence, the application logic can be deployed in a community as simple stateless service.

Distributed services in nature can also be supported in communities. These services are composed by a set of different components that interact with each other through the network to provide a more complex service. They employ the resources of the community and take advantage of resource discovery and aggregation features and the replica management mechanisms. However, they implement their own inter-component communication and replica management mechanisms.

Distributed or three-tier services with multiple components can be referred as well as *composite services* built from simpler parts. These parts or components, as in the pure distributed services, would create an overlay and interact with each other to offer more complex services, but should be managed by the community mechanisms instead of being the service itself who internally handles all the communication and replica matters. As an example, each of the four components in Figure 1.4 can be replicated to enhance the service performance.

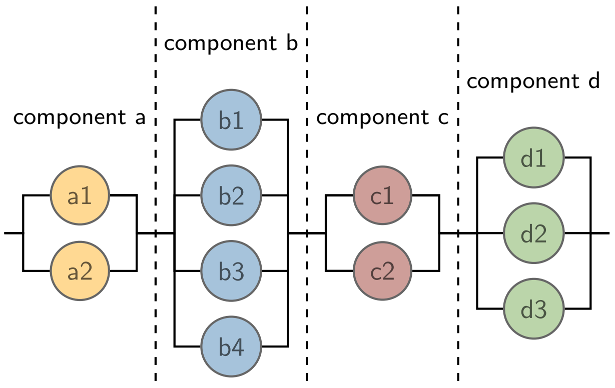

Figure 1.4: Composite service representation

Service deployment

Independently of the actual implementation, the service deployment process in the contributory computing model would generally proceed as shown in Figure 1.5.

A user willing to deploy a service should send its specifications to a designated responsible for that service. For example, on the one hand CoDeS relies on DHT facilities to determine

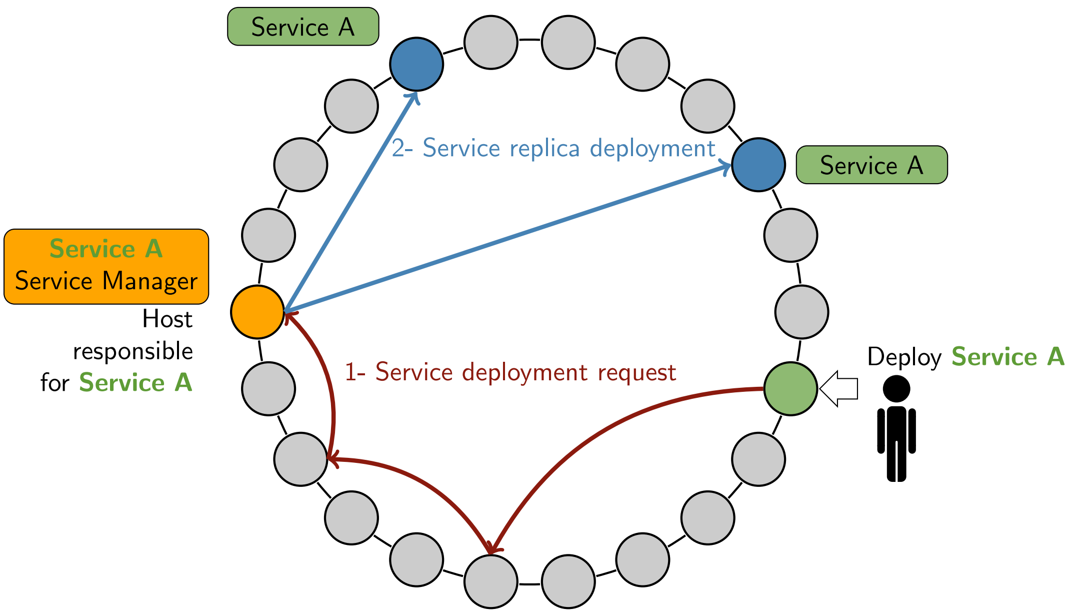

Figure 1.5: Service deployment process in the contributory computing model

the responsible of each service (labeled as *Service Manager* in Figure 1.5) and route the requests to her. On the other hand, Nebula relies on a central server to coordinate all the contributed resources and deploy replicas. In this example, a user requesting a service composed by two identical replicas contacts the node responsible for this new service through a DHT-based overlay. This host then determines where to allocate the needed replicas and remotely instantiate them in the chosen nodes.

On a service deployment request, the *Service Manager* must decide where to allocate the desired number of replicas specified by the user demanding the service. While in CoDeS the middleware maintains a list of available resources and the provided tools enable feature selection when selecting nodes, in Nebulas the coordination entity perform all these actions in a centralized manner.

1.4 Allocation methods

We pointed out the need of a strategy to select a set of hosts among a larger pool to create a service overlay in contributory systems. With this purpose, many selecting approaches would be valid: random selection, greedy methods, exact methodologies or heuristics/metaheuristics.

In general, dealing with Internet distributed systems often involves a large amount of highly scattered nodes. The scale of contributory platforms is directly related to the number of participants in the community, their differences contribute to increase the heterogeneity and their stochastic behavior increases the system dynamism. In consequence, the study of these systems often becomes an NP-hard or NP-complete problem [45].

The resolution of NP-hard or NP-complete problems is a deeply studied research field. Most of the existing proposals could be classified into exact methods or heuristic/metaheuristic approaches. On the one hand, the former try to find the optimal solution to the problem, regardless the computational time it takes. On the other hand, the latter define steps or methodologies to follow in order to obtain *good* quality solutions (often close to the optimal value for the problem) in restrained computing times [46].

Due to the large-scale nature of the contributory platforms and the node's dynamism and stochastic behavior, the methodologies presented in this thesis will focus on heuristic approaches that provide high quality host selections (regarding a given parameter) in reasonably moderate execution times. We expect this methods to be used in real system deployments, in which is desirable to provide a user-interactive service deployment process. This means it would be tolerable for the system to spend few minutes determining which is the best allocation for a given service deployment request, but it would not be acceptable to spend hours until the service is finally accessible.

1.5 Objectives and scope

While contributory communities aimed at creating a platform to deploy general purpose applications and long-lived services, Nebulas focused on data-intensive applications. Both proposals rely on non-dedicated computing resources as their major strength. Treating both systems as regular cloud platforms, in which most of the resources are well-maintained and well-provisioned, would lead to inefficiency, resource waste, service malfunction and poor quality perception from the final users.

Due to the natural dynamism and heterogeneity of user-contributed systems and the stochastic behavior of their owners, it is imperative to devote efforts to determine resources' roles into the deployed services. Minimizing the number of utilized resources to efficiently perform a task or service would result in more scalable distributed systems.

The main goal of this thesis is to **optimize the resource selection for service allocation in the contributory computing model**. More precisely, we focused on different replica allocation methodologies taking account different parameters, aiming to enhance system performance and user quality perception. Therefore, we studied optimization techniques in different directions, so the quality of these systems could be improved.

Availability-aware service allocation Providing high availability services on top of non-dedicated computers is a strong research challenge and is one of the main strengths of popular Internet applications. If achieved, it would be possible to attract services and its users from current cloud infrastructures to community-based platforms. The heterogeneity on the hosts involved in such environments and the topological complexity of advanced Internet applications hinders the prediction of overall service availability. We believe that using historical availability behavior to select nodes in the system would lead to higher service availability levels. Thus, user impression is improved and more users and services could be attracted to contributory environments.

Energy-aware service allocation Current trend on computing research study the environmental issues caused by computational infrastructures. The paradigm of *green computing* is an umbrella that covers topics from unrelated disciplines [47], including hardware novelties, energy-aware scheduling algorithms or innovative cooling systems among others. The idea of system energy consumption reduction fits naturally in the contributory computing paradigm. Seizing under-utilized resources that would be still consuming energy if they were not in use is an improvement *per se*, but dispersion also plays an important role to avoid expensive cooling systems. Yet, proper assignation of components or tasks to hosts with lower energy consumption would have a direct impact on the reduction of the power consumption of contributory platforms.

Different hardware and software means different energy consumption models for each of the involved resources. Taking into account these energy models, the load of each computer at a given time and the requirements of the services, it is possible to forecast the power consumption a service would generate when hosted on different hosts. Combining this with the availability-aware node selection, we can determine the subset of nodes to host a service that guarantees a given level of availability with a lower energy consumption.

Related with the environmental concerns about computing infrastructures, scheduling in computing is one of the most critical aspects when planning independent tasks executions in a set of computers, dealing with either dedicated or non-dedicated resources. Current proposals consider the execution times to be deterministic, which can be far from reality. Operative system interruptions, network congestion and storage or network delays can have a great impact on these execution times. Hence, reducing the idle or queuing times for certain tasks would lead to less power consumption.

Network-aware service allocation Resource dispersion in contributory scenarios can be a drawback or an advantage, as the Nebula authors claimed in their research. Multi-component services create an overlay that abstract from the actual underlying network connections. Withal, those services requiring intensive inter-component communications to perform their intended tasks would perform better if the replicas are placed close to each other in the underlying network. On the contrary, delay- or *jitter*-sensitive or bandwidth-intensive services should perform much better if their replicas are as close as possible to their final consumers.

Relying exclusively on user-contributed resources disables globally planned long-term deployments. Still, if some information of the underlying-network and host location is available, decisions can be made to strengthen certain types of applications and benefit the community. Again, appropriate selection of nodes with certain properties would be profitable for the quality of service and convenient for a sensible resource usage.

1.6 Original contributions

In the process of achieving the described objectives above, this thesis came up with the main research contributions explained in the following sections.

1. An availability-aware mechanism for composite service allocation

We addressed the challenge of guaranteeing service availability for composite services over non-dedicated resources by designing and implementing a *simheuristic* that combines a metaheuristic with Discrete Event Simulation (DES) into its operation. We created a framework that seeks for service deployments that guarantee a certain availability requirement.

We first built a simulation tool to predict global service availability involving hosts with different availability behaviors. Thanks to this tool, we can forecast the availability of composite services with complex topologies and composed of nodes with very different behaviors.

We devised a new parameter-less metaheuristic that defines a simheuristic framework [48] to select a set of nodes to support a given service with a certain availability requirements, taking advantage of the availability simulator and an Iterated Local Search framework [49]. The novelty of our proposal is twofold. First the combination of simulation techniques inside of a metaheuristic eliminates some of the restrictions of similar proposals in terms of system topology, component behavior or scale of the system. Second, its zero-parameter configuration makes it suitable for very different conditions.

Moreover, comparing our simheuristic with other state-of-the-art proposals, our mechanism obtains better results in less computation times, which makes it suitable for user-interactive service deployment processes.

As a side contribution to our availability-aware methodology, we designed a distributed application based in the contributory computing paradigm to offer a microblogging service (similar to Twitter) to community users.

We envisioned a theoretical usage model from Twitter statistics and validated by simulation the feasibility of our proposal in different usage scenarios, regarding bandwidth and storage requirements for community supporters. With this real application, we expect to clarify the kind of services targeted by our proposal and to identify easily where our other contributions would fit in the contributory model. After its validation, we applied the availability-aware allocation mechanism explained above to an artificially simulated community supporting the distributed microblogging service to prove its value in a real application.

2. A simheuristic approach to schedule distributed task execution

We designed a methodology to schedule tasks in a distributed environment considering deviations in running times, which is a more accurate approach to reality.

We modeled the task scheduling scenario as a Flow-Shop Problem [50] with Stochastic Times and proposed a simulation-optimization methodology to exploit current state-of-the-art heuristics to tackle this problem. Our contribution was to bring the simulation-optimization methodology to the distributed computing arena, so task planning becomes more realistic.

The proposed methodology obtains significantly lower overall computing times than random task processing when combined with two well-known heuristics for the Flow-Shop Problem. Our assumption was that shorter makespan for overall execution times would result in lower energy consumption globally. We ascertained that by simulations with well-known benchmarks for the FSP and typical energy consumption from present datacenter servers.

3. A combined availability- and energy-aware service allocation mechanism

We reviewed in depth the literature on replica allocation mechanisms that considered the energy consumption in the decision process. This study helped us to discuss the challenges that the green computing topic arises to the Operations Research community applied to the distributed computing field.

We present an extension of our availability-aware allocation mechanism to determine which is the subset of hosts that consumes the minimum power while maintaining a certain level of service availability.

We considered the energy to be a stochastic variable with high variability over time. We built a simulator of energy consumption of different hosts from a given energy model for each one. We then compared the service allocations obtained from a greedy host selection process and from our methodology and discovered that significant reductions on consumption can be made with small degradation of the service availability.

4. A network-aware mechanism for service allocation

We designed a methodology to allocate service replicas in a network graph aiming to minimize the distances from each client to its closest service replica.

We modeled this problem as a Facility Location Problem [51, 52], on which service replicas are facilities (with a set up cost) and the clients in the network are customers. Then, the goal is to minimize the sum of distances and set up costs of the selected replicas.

We assessed our method with the classical benchmarks for the FLP and then validated our approach in a real network topology fom a community network, showing the goodness of our proposal on finding closer replica locations to all clients in the network.

1.7 Dissertation outline

The rest of this dissertation is organized in four more chapters, which cover the contributions of this thesis.

Chapter 2 is focused on describing the availability-aware mechanism that combines computer simulation with an state-of-the-art metaheuristic to find the most suitable subset of nodes in a contributory platform to host a composite service.

Chapter 3 reviews several energy consumption matters of distributed computing systems in general. After that, it presents a methodology to schedule ordered tasks in a set of servers and another methodology that guarantees a certain service availability for contributory services while reduces the energy consumption of these systems.

Chapter 4 presents a network-aware methodology to allocate services in a contributory platform in such a way the network distances of all potential clients to their closest replicas is minimized.

Chapter 5 reviews the main conclusions of this thesis and discusses the contributed results. It also draws several lines for future research works that could continue this thesis.

2 Availability-aware allocation for composite services in contributory environments

This chapter describes an availability-aware mechanism for contributory computing platforms. This procedure is based on a simheuristic that combines an Iterated Local Search heuristic with a complex system availability simulator.
The contents of this chapter were extracted from publications [P1, P3, P7, P8, P9, P10, P12].

2.1 Overview

As previously evidenced in Chapter 1, there is a promising trend to promote the utilization of non-dedicated resources for offering cloud-like services over the Internet. Despite the clear benefits of sharing and aggregating idle resources for deploying Internet services in a community of users, the fact that these are non-dedicated and very heterogeneous resources makes them extremely stochastic and unpredictable. Therefore, complex mechanisms — mainly based on the use of costly redundancies— must be developed in order to guarantee reasonable availability levels for the service.

In this particular context, we generally understand the availability of an individual resource or a system of resources as the probability that it will be operative, at a given time, when working under specified environmental conditions [53]. In distributed computing, availability is understood as the fact a host has a working connection to the network and is reachable by other nodes [54]. Also, the hardware must be on and operational and the associated software must be in a state that can respond to remote requests [55].

Accordingly, we propose in this chapter a novel simheuristic to generate cost-efficient configurations of non-dedicated resources able to support Internet services (as described in Section 1.3.2) with a high availability level. Our methodology deals with the stochastic optimization problem of determining a minimum-cost configuration of non-dedicated resources able to support a service while maintaining the service availability level over a user-defined threshold.

The main idea behind our approach is to design a metaheuristic algorithm which, starting from a feasible but costly solution, performs an oriented local search trying to replace expensive resources by cheaper ones —usually offering somewhat lower availability levels. For each new configuration generated in this iterative process, simulation is employed to estimate the new global availability of the service. This estimation is then used to check if the new and less expensive configuration offers an availability level higher than the one specified by the user. As long as more time is available, the algorithm iterates this search process looking for cheaper configurations of resources that offer service availability levels above a user-defined threshold.

Despite its relevance in future Internet computing practices, this problem has not been specifically analyzed so far. Efficient approaches have been developed for solving similar problems in the case of grid and cloud computing, though many of these approaches assume highly-dedicated resources, which are characterized by high availability levels. Other restrictive assumptions typically refer to the system size, the system topology —i.e., the logical interconnections among the components offering a computing task or a service—, and the probability distributions employed to model the failure and repair times of each resource.

2.2 Problem description

We defined the concept of service earlier in Section 1.3.2. Basically, a composite distributed service can be seen as a system of interrelated components that must be completed, over a time period, by a set of different resources.

In terms of availability, a composite service will be available (operative) for the users if some logical conditions are satisfied. Thus, at any time there must be some nodes offering each of the service critical tasks. The logical conditions define an underlying logical topology for the service to be operative. This topology can be represented as a graph or network of vertexes (resources completing specific tasks) and edges (logical interrelationships among tasks), The example in Figure 2.1 shows five components supported by seven hosts, being nodes 2 and 3 and nodes 5 and 6 replicas of the same component.

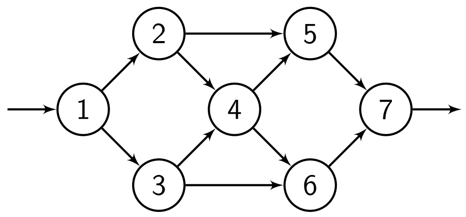

Figure 2.1: Graph representation of a composite distributed service

Notice the availability of these services will depend upon (a) the individual availability of the resources selected to complete each task; and (b) the underlying topology defining the conditions for the service to be operative. The service availability level could be trivially increased by employing those resources with the highest availability levels, but these resources are likely to be the most desired and expensive ones as well.

The service availability level could also be increased by adding redundancies to the underlying topology, i.e., by assigning a large number of redundant resources to each component.

However, this policy would also increase the cost of deploying the service and could limit its scalability. Thus, our goal is to find a configuration of nodes which delivers the service with a high availability and at the lowest possible cost.

More formally, let $G = (\mathcal{N}, \mathcal{E})$ be a graph representation of the service, where $\mathcal{N} = \{n_1, n_2, ..., n_n\}$ denotes the set of n nodes (resources) that can be used to deploy it, and $\mathcal{E} = \{e_1, e_2, ..., e_m\}$ denotes the set of p minimal paths defining the logical topology of the service [53]. In other words, the service will be available if, and only if, at least one of these minimal paths is fully operative —at any given time, t, a minimal path will be operative if, and only if, all the nodes in it are operative at t. For each node n_i $(i = 1, 2, ..., n)$, we will consider the cost of using it, $c_i \geq 0$, its current status at time t, $s_i(t)$, as well as a binary variable x_i. The variable x_i will take the value 1 if node i is selected to deploy the service, and 0 otherwise.

Similarly, the status of node i at any given time t, $s_i(t)$, will be either 1 if the node is operative at t, or 0 if it is not. Remark that $s_i(t)$ will depend upon the behavior of two random variables, F_i and R_i, which represent *failure* times (time to the next disconnection) and *repair* times (time to the next connection) for node i, respectively. Weibull distributions are usually employed to model these random variables, but other distributions like the Exponential or the Log-normal are also frequently used.

Finally, we denote the availability of the service at any time t by $A(t)$. As discussed before, $A(t)$ will be a function $g()$ of both $\mathcal{E}$ and $s(t) = (s_1(t), s_2(t), ..., s_n(t))$. Then, our goal will be to find a subset of nodes $\mathcal{M}$ in $\mathcal{N}$ such that the total costs of the resources are minimized while the service availability at any time t, $A(t)$, is greater than a user-given threshold, A_0, for all t in the interval $(0, \mathcal{T})$. This problem can be expressed as in Equation 2.1.

$$
\begin{aligned}
&\text{Find} && \mathcal{D} : \mathcal{N} \to \mathcal{M} \\
&\text{that minimizes} && f(\mathcal{D}) = \sum_{\forall i \in \mathcal{N}} x_i c_i \\
&\text{subject to} && s_i(t) \in \{0,1\}, \forall i \in \mathcal{N}, \forall t \in (0, \mathcal{T}) \\
& && s_i(0) = 1, \forall i \in \mathcal{N} \\
& && F_i \approx Failure(\alpha_i, \beta_i), \forall i \in \mathcal{N} \\
& && R_i \approx Repair(\gamma_i, \delta_i), \forall i \in \mathcal{N} \\
& && s_i(t) = f(F_i, R_i), \forall i \in \mathcal{N}, \forall t \in (0, \mathcal{T}) \\
& && x_i \in \{0,1\}, \forall i \in \mathcal{N} \\
& && c_i \geq 0, \forall i \in \mathcal{N} \\
& && \bar{A}_{\mathcal{D}} = g(\mathcal{E}, s(t)) \geq A_0, \forall t \in (0, \mathcal{T})
\end{aligned}
\tag{2.1}
$$

Our problem can be formulated as well as a discrete optimization problem, namely:

Problem 1 *Find the deployment $\mathcal{D} : \mathcal{N} \to \mathcal{M}$ that minimizes the objective function $f(\mathcal{D}) = \sum_{\forall i \in \mathcal{N}} x_i \cdot c_i$, subject to the constraint $\bar{A}_D \geq A_0$.*

From the formulation above, we first need to solve the issue of computing the service availability, $A_{\mathcal{D}}(t)$, for different nodes-topology configurations. As discussed earlier, it seems reasonable to assume that we have information about the availability behavior of each node —the associated probability distributions can be obtained by fitting historical data on each resource. Similarly, it seems reasonable to assume that the service topology can be expressed as a series of minimal paths. However, depending on the specific distributions and topologies, it might not be easy or even possible to compute the service availability by using exact methods.

Hence, in most realistic scenarios —characterized by non-normal distributions and network topologies with hundreds or thousands of nodes—, algorithms based on discrete-event simulation become an excellent alternative to exact or analytic methods [56, 57], which often can not easily represent factors such as degradation, repair processes, obsolescence and component replacement. They can provide estimates for the service availability without having to consider specific assumptions on the probability distributions modeling individual failure and repair times or on the service topology. Our simulation-optimization approach makes use of such an algorithm.

2.2.1 Complexity of the problem

If $|\mathcal{N}| = n$ and $|\mathcal{M}| = m$, then the size of the search space is the number of different deployments, $n(n-1)\cdots(n-m+1) = n!/m!$. As we mainly deal with very large-scale systems, typically $n \gg m$, which makes $\frac{n!}{m!}$ very large, and rules out any form of brute-force search.

Notice that the different processes will have a certain priority within the service, generally speaking. Although not strictly required, we prefer to respect the relative priorities of the processes during deployment, i.e. we assign critical component replicas to nodes with higher availability. This enables us to reduce the search space, since now we just have to look for a subset of nodes $\mathcal{S}$, with cardinality m. Once we fix the subset $\mathcal{S}$, the most critical component is assigned to the node of highest availability, the second most critical component is assigned to the node with the second-highest availability, and so on. In other words, $\mathcal{D}$ is uniquely determined by $\mathcal{S}$, provided that the process priorities are all different, and the node availabilities are all different. The size of the search space now becomes $\frac{n!}{m!(n-m)!}$.

The most important benefit of this restriction is that our problem reduces to a variant of a well-known combinatorial optimization problem. Note that the objective function f can now be reformulated as a function of $\mathcal{S}$, instead of $\mathcal{D}$. Next, we shall introduce new variables, in order to reformulate the problem as a maximization problem.

Let us assume that the cost of a node is bounded above by MaxCost, and let us define the *profit* of a node n_j as $\text{MaxCost} - c_j$. The total profit of a deployment is obviously defined as $f'(\mathcal{S}) = \sum_{\forall j \in \mathcal{S}} (\text{MaxCost} - c_j) = \sum_{\forall i \in \mathcal{N}} x_i (\text{MaxCost} - c_i)$. Let us also define the *weight* of n_j as $1 - \bar{a}_j$, and the overall weight of the deployment $\mathcal{S}$ as $1 - \bar{A}_{\mathcal{S}}$. With these new definitions, our Problem 1 can be restated as:

Problem 2 *Find a subset of nodes $\mathcal{S}$, with cardinality at most m, that maximizes the overall profit $f'(\mathcal{S})$, subject to the constraint* $1 - \bar{A}_{\mathcal{D}} \leq 1 - A_0$.

This turns out to be a variant of the well-known *k-item Knapsack Problem* (kKP) [58, 59]. The only difference with the k-item Knapsack Problem is that in our case, $1 - \bar{A}_D$ is not necessarily a linear function of the individual weights $1 - \bar{a}_j$. Being a generalization of the k-item Knapsack Problem, we can state our problem is clearly NP-hard. Thus, as we already discussed in Section 1.4, heuristic approaches can be developed for obtaining good solutions to this complex problem when dealing with large-scale scenarios as the ones intended in contributory computing.

2.3 Related work

Many studies focused on availability studies of distributed computing environments. More recently, some of them focused on those based on heterogeneous and non-dedicated resources

One of the first studies discussing availability of distributed (transactional) systems was done by Triantafillou [60]. He argued for finding high levels of availability without increasing costs unnecessarily. Also, he addressed the need to investigate availability in large-scale heterogeneous systems. Other authors also discussed availability in distributed systems, proposing two-level hierarchical models to analyze these issues [61]. At the system level, they use a graph-based approach to analyze the availability of distributed programs. On the other hand, they use Markov models for predicting availability at the component level.

Due to the growing importance of distributed environments based on heterogeneous and non-dedicated resources —e.g. peer-to-peer and volunteer-computing networks—, there have been many recent studies about node-level availability in such systems [62, 63, 64].

Other studies focused on studying CPU availability rather than host availability, a similar but more restrictive approach on which load [65], user working periods or OS interruptions were considered [66]. Thus, the host is accounted as available only when it is actually available to execute tasks in a volunteer computing platform. In [66], the authors studied the execution availability over four different traces from desktop and laboratory computers and a dedicated cluster. They extracted some per-host and aggregated statistical information that clearly showed the heterogeneity and volatility of desktop grid resources. After defining this host characterization, the authors proposed a performance model for different granularity task execution in distributed systems. Finally, they compared the performance of executing tasks in a dedicated cluster environment and in desktop grid resources, finding a relation between both (called cluster equivalence).

Bhagwan et al. analyzed the peer-to-peer sharing system used in Overnet and argued for developing new methods to test the availability of systems subject to complex temporal variability [54]. They argued that the availability of hosts in these scenarios should be modeled by a combination of two distinct probability distributions: (a) distributions reflecting the daily on-line/off-line status of hosts, and (b) distributions reflecting the entrance of new hosts to and permanent departures of old hosts from the system. Regarding the latter, their results showed that each day over 20% of the hosts in Overnet were new to the system.

Douceur was one of the first who examined the host availability in large-scale and highly-distributed computing systems [55]. He studied four traces from Microsoft enterprise desktop computers, Internet desktop computers, Gnutella hosts and Napster hosts, founding several common traits in their behavior. He then proposed a graphical, mathematical and behavioral model for the host conduct and discovered a graduated mix of two uniform distributions. This could be caused by the existence of two types of hosts, identified by the author as one group with cyclical behavior and another which are left on at all times.

An interesting description of availability in distributed systems can be extracted from this same article by Douceur [55]. In this context, the term availability refers to everything needed for a host to be seen by another host: the hardware must be on and operational, the associated software must be in a state that can respond to remote requests and a connection must exist between the two hosts.

Giesecke et al. also discussed ideas related to reliability and availability of peer-to-peer systems [67]. Using real data to build their model, they developed a discrete event simulator for peer-to-peer systems. This simulator evaluated different replication strategies based on voting policies like ROWA and majority consensus.

According to Mickens and Noble [68], achieving efficiency in a peer-to-peer system requires the analysis of system availability. They argued that previous peer-to-peer models for system availability provide estimates of uptime, but cannot predict changes in availability over time. Therefore, they proposed new methods for predicting availability based on signal analysis techniques and information theory. They also described several useful applications of availability prediction in distributed systems: reducing bandwidth consumption, increasing data availability and reducing message latency.

All these studies have provided valuable information about the availability patterns of several types of computational resources included in different computing paradigms. However, they focus mainly on node-level availability instead of on service-level availability. In particular, our methodology aims at efficiently deploying composite services, formed by a number of interconnected components, on a set of intermittently available resources. Despite its relevance for practical applications of contributory communities, to the best of our knowledge no previous proposals have addressed this problem so far.

The most similar issues that have been previously addressed in the literature are those related to: (a) providing efficient levels of *collective availability* to a service or to data in a distributed storage system; and (b) *workflow scheduling*. A literature review on these two issues, together with remarks on the differences and original contributions of our approach, are included next.

2.3.1 Service availability

Andrzejak et al. introduced the notion of collective availability referred to making a service or a data object available through replicating it in a number of machines [69]. The service is considered to be available if at least k out of n computers where the service is located are available. This is strongly related to some fitting models for file storage.

In effect, file objects can be stored using erasure codes [70], which divide a file object into n fragments. By doing so, a client only needs to obtain k fragments ($k < n$) to restore the entire file. Many distributed storage systems have tried to guarantee collective availability for data objects [71, 72]. Bhagwan et al. argued that managing availability in large-scale dynamical systems is a crucial but complex task in achieving system efficiency [71]. For their predictions, they proposed a peer-to-peer architecture in which the storage system could predict the future availability of components, determine the appropriate level of redundancy, and automatically initiate repair actions according to users storage needs. Pàmies et al. studied the data redundancy costs and modeled the relationship between data redundancy, host availability and retrieval times in distributed storage systems [6]. He devised new procedures to optimize the data assigning process in heterogeneous storage systems and he proposed a

PSO to determine the best way to assign an amount of redundancy to a set of nodes so that data availability is maximized. The author presented a data assignment mechanism that assigns to each node an amount of data proportional to its availability, maximizing the overall storage capacity.

The case of guaranteeing collective availability for computational services has also been studied in some previous works [69, 73, 32]. In these models, a service deployment usually implies considering n identical replicas of the service. A k-out-of-n topology is then considered, meaning that the service will be available whenever at least k replicas are available, independently of which ones. Many real-life services do not fit this model, since they present different inter-component connections and system topologies. Therefore, a flexible simulation-based methodology, like the one presented in this chapter, can be really helpful in such cases.

Andrzejak et al. tried to identify and evaluate the availability of a set of resources in large-scale distributed systems, describing collective availability [69]. This approach employs a virtualization layer to mask any lack of availability in individual resources. The researchers tested a number of prediction methods, investigated factors influencing prediction error, and identified a number of indicators that predict the future availability of resources. Also, they showed how these methods could be used to predict the availability and associated costs of resource groups. They later expanded this work to discuss using a mixture of dedicated (with high levels of availability) and non-dedicated (very volatile) hosts to ensure efficient web services in large-scale distributed systems [73]. Their approach includes ranking non-dedicated nodes based on their past behavior, modeling short-term availability, and simulating availability levels of groups of nodes. In fact, actual data for 10,000 non-dedicated nodes was used.

Lázaro et al. analyzed a large set of host availability traces from SETI@Home project aiming to identify and classify different types of hosts and quantify their contribution to overall system availability on the long run [32]. Then, they presented an availability prediction tool based on bit vectors standing for a whole week host availability. They apply this predictor to determine sets of hosts with negatively correlated availability patterns and so deploy reliable services on top of them, trying to minimize file transfers for the service binary files. They assessed their method by trace-driven simulations and showed it generates service deployments with lower redundancy but higher collective availability.

The problem of service availability is also addressed in [74], where Dai et al. develop a Markov model to evaluate the availability of resources in a grid. Their goal is to minimize the cost of grid systems by determining the optimal number of (homogeneous and dedicated) servers to be employed. However, while their model takes into account server load issues, it also makes some strong assumptions. Specifically, they assume parallel-like topologies, exponential failure and repair times and independence of failures among different resources.

Kondo et al. measured and characterized time-dependent dynamics of real large-scale distributed systems (in particular BOINC, with more than 110,000 hosts) [75]. They focused on identifying correlated availability patterns. They used different metrics along with clustering techniques to automatically detect meaningful patterns. In this way, they identified groups of resources with correlated availability and discussed how correlated clusters of resources can be used to improve the efficiency of volunteer computing.

These previous works show that using actual data from real systems is vital in understanding availability in distributed systems. For this reason, Kondo et al. created the Failure Trace Archive [76], a repository of availability traces for a variety of real distributed systems. These traces are presented in a common format that enables comparative studies. Moreover, they modeled host availability in nine distributed systems, including PlanetLab, Overnet and Skype, among others. The repository also includes a SETI@home trace of more than one year and comprising 230,000 hosts. This last trace was used by Javadi et al. to validate an efficient method for discovering subgroups of hosts with similar statistical availability [77]. Of course, subgroups can more accurately be modeled with similar probability distributions. According to the authors, at least 21% of the nodes in SETI@home can be accurately modeled by theoretical distributions.

Finally, many of these proposals assumed some historical knowledge about host availability intervals, but none presented a method to obtain this information in a distributed fashion on a working system. Morales and Gupta developed AVMON [78], a system that allows host availability monitoring in large-scale distributed systems. They discuss several distributed applications that employ nodes which continuously arrive and depart from the system. AVMON is presented as a solution to monitor availability not only in regular systems but also in those with *selfish* or *corporate* nodes. That is, AVMON deals with those nodes that manipulate their and others' availability for personal profit.

2.3.2 Workflow scheduling

The workflow scheduling problem has been exhaustively studied in the field of grid computing, in which a group of machines have to be selected to efficiently deploy a set of interconnected tasks and guarantee its finalization. It is similar to the problem we address in that a group of machines have to be selected to efficiently deploy a set of interconnected tasks. The availability levels of the selected hosts contribute to guarantee the correct finalization of the execution process. However, in our case the goal is not necessarily to guarantee the finalization of a time-bound set of tasks, but to keep a long-lived service available during an undefined period of time. In the remainder of this section we review some of the existing literature for workflow scheduling in grids and discuss how our approach differs from them.

In [79, 80], the authors address the optimization problem of task allocation in heterogeneous distributed systems. Their goal is to maximize system reliability. Attiya et al. propose a simulated annealing algorithm to solve the aforementioned optimization problem [79]. In their approach, however, they assume the system follows a series-like bus topology and that it is fully connected. They also assume that all resources have the same cost. In [80], Kang et al. use an approach based on a honeybee mating optimization algorithm. In their experiments, they use a uniform distribution to model node-level reliability, and only small-scale systems (up to 8 processors) are analyzed. Our approach makes less restrictive assumptions, and it can be used for dealing with more complex, heterogeneous, and larger system topologies.

In [81], Tao et al. propose a Markov Chain model to predict future availability of grid nodes without adding significant overhead to the system. Their work is mainly oriented to predict node-level availability in order to minimize the tasks completion time (also known as makespan). Therefore, they do not consider system-level availability, which makes their goal quite different to the one introduced in our research.

In [82, 83, 84], the authors present a particle swarm optimization algorithm to solve a multi-objective task allocation problem in distributed computing systems. They mention several optimization objectives that can be considered while assigning tasks to resources in a distributed system. Among these potential objectives, they cite (a) minimization of execution and communication costs; (b) maximization of system reliability and safety; and

(c) maximization of fault tolerance throughout redundancy. However, their computational experiments are run over small-scale topologies (up to 9 processors).

In [85], Tang et al. develop a reliability-aware task scheduling algorithm in which interdependencies among tasks are considered. They show that the performance of their algorithm surpasses that of previous algorithms not considering reliability. However, some of the assumptions they use (e.g., independent failures following a Poisson process) restrict the applicability of their model. Our approach tries to be more flexible in the sense that it does not need to make these restrictive assumptions.

Yu et al. [86] propose a cost-based workflow scheduling algorithm that minimizes execution cost while meeting the deadline for delivering results over Utility Grids. Users are required to pay in the system, hence cost reduction becomes a key factor. In [87], Zeng et al. are concerned on guaranteeing user requirements for deployment of services over reliable resources. However, they do not consider the case of using unreliable (non-dedicated) resources. Finally, Brandicet al. try to bind the tasks in a workflow so that grid services satisfy user requirements [88]. However, they do not consider system efficiency (costs), which is a fundamental part of our approach.

2.4 A simheuristic approach

In this section we propose a simheuristic to address to the problem of deploying Internet services over non-dedicated resources with relatively low availability levels in large-scale systems. Our proposal integrates discrete-event simulation into a metaheuristic search framework. The goal is to find a solution which delivers the service with a certain probability and, at the same time, employs a low-cost assignment of resources for efficient work execution.

First, we consider a service deployment process as the one explained earlier in Section 1.3.2 and sketched in Figure 1.5: a user requests a service deployment, a service manager determines the most suitable set of nodes to support the service with our proposed methodology and finally the service is deployed in the chosen hosts. The basic steps of our approach are described next and are also sketched in a flowchart in Figure 2.2:

1. **A user requests to deploy a service:** a user wants to deploy a service with a set of requirements, specifically:

- a clear description of the service, including the different agents that must deploy it as well as their specific roles;
- when appropriate, computational requirements (CPU, RAM, geographic location, quality of its Internet connection, etc.) for each service agent;
- the time during which the service must be operative or, alternatively, conditions that imply the end of the service;
- the conditions that must be satisfied to consider the service as successfully deployed and executed (i.e., the logical topology related to the service availability);
- a lower bound or threshold for the availability level that the system must attain, i.e., a lower bound for the probability that the service can be satisfactorily deployed and executed (e.g.: 0.95, 0.99, etc.); and
- when resources have an explicit cost, the available budget for booking computational resources (nodes).

2. **The service manager proposes an initial feasible solution:** the host designated as the service responsible determines an initial solution by selecting a subset of free nodes among those resources with the highest availability levels —which are likely to be among the most expensive ones as well. To do so the manager must have information about:

   - which resources are idle and their cost;
   - the computational characteristics (CPU, RAM, geographic location, quality of its Internet connection, etc.) of each resource in the system;
   - the availability behavior of each individual resource, i.e., it is assumed that we know the probability distributions modeling the failure (disconnection) times and repair (recovery) times for each resource.

   Notice also that during the initial selection of nodes we could consider policies that promote independence among failure and repair times of different nodes —e.g., by selecting geographically dispersed nodes or nodes belonging to different clusters inside the community. Once the initial solution has been constructed using highly-available resources, the manager can check for service feasibility. In particular, the manager must estimate the system availability function to ensure that it satisfies the user's requirements. This estimation can be performed using simulation-based algorithms, as explained in next section.

By construction, we expect this initial solution to be feasible (i.e., to offer a higher availability level than the minimum requested by the user), although quite expensive —since it has been built using expensive resources. Depending on the user's budget and on the maximum time allowed for providing a solution, the manager can then start a heuristic Iterated Local Search [49] process to look for cheaper alternatives without violating the availability threshold. This iterative process is presented in step 3.

3. **The service manager improves the solution:** to maintain the availability level of the service above the lower bound value defined by the user, we identify the critical nodes —from an availability point of view— in the current solution. Thus, the manager can start a local search process by substituting expensive and noncritical nodes with cheaper resources. It can also try replacing critical and expensive nodes with redundant structures including cheaper resources. By introducing redundancy, the manager may get the same availability level with a lower cost.

   At this point, Discrete-Event Simulation (DES) can be used again to find the defining characteristics of this modified system. One of the simplest rules to follow during this iterative process could be the following one: if the new solution is still feasible and also cheaper than the previous solution, then replace the old solution with the new one; otherwise ignore the new solution and look for another one. Of course, several meta-heuristic frameworks —like the one explained in the next section— can be developed to perform more advanced local search processes able to provide pseudo-optimal solutions. In particular, the approach employed here also takes into account that in some real-life scenarios solutions might be needed in just a few seconds or minutes.

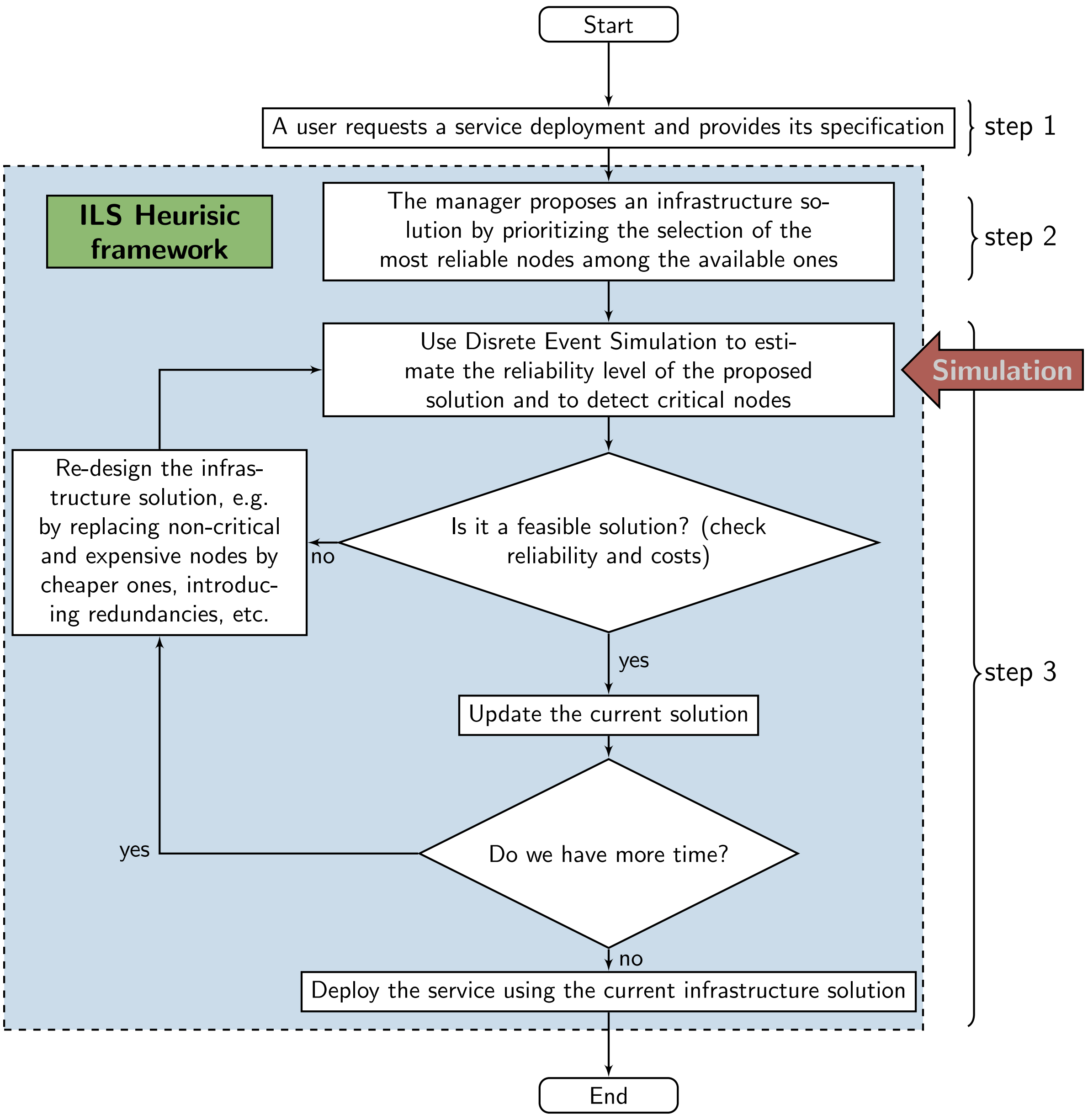
Start
A user requests a service deployment and provides its specification
step 1
ILS Heurisic framework
The manager proposes an infrastructure solution by prioritizing the selection of the most reliable nodes among the available ones
step 2
Use Disrete Event Simulation to estimate the reliability level of the proposed solution and to detect critical nodes
Simulation
Re-design the infrastructure solution, e.g. by replacing non-critical and expensive nodes by cheaper ones, introducing redundancies, etc.
no
Is it a feasible solution? (check reliability and costs)
yes
Update the current solution
yes
Do we have more time?
no
Deploy the service using the current infrastructure solution
step 3
End

Figure 2.2: Basic flow diagram describing our simheuristic methodology

2.4.1 A discrete-event simulation algorithm for complex system availability estimation

This section reviews the basic ideas behind our discrete-event simulation algorithm used by our simheuristic framework to estimate the availability of each new nodes-topology configuration. The simulation component of the integrated approach presented in this chapter is an adapted version of the algorithm introduced in [89].

For simplicity, we will only consider two possible states for a node: available (i.e., operative) or not. We call a transition from operative to non-operative a *failure* or *disconnection*. A transition in the opposite direction is a *recovery* or *repair*.

The availability simulation algorithm assumes that probability distributions modeling availability behavior of each individual resource are known. Since these will be distributions modeling positive values (time to failure or time to repair), Weibull or Log-Normal probability models will be typically preferred over Normal probability models. The algorithm also assumes that the system topology can be represented as a set of minimal paths [53, Section 1.3.2]. We depict in Figure 2.3 an example of a topology graph decomposition into its set of minimal paths.

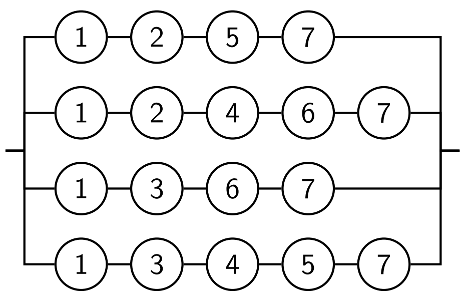

Figure 2.3: Minimal path decomposition for the system in Figure 2.1

The algorithm uses discrete-event simulation to generate an artificial life-cycle for the entire service. A system status shift schedule, represented as a list sorted by time and containing the events that occur (failures or recoveries) is built at this step. Failures and recoveries will be considered as *complementary* events. We begin by randomly generating failure times for each node based on its associated failure distribution. When a new event (component failure or repair) occurs, we add this event to our list and schedule a complementary event for the same component.

Assume that we need the service to be functional for a time *simTime*. Then, for each event that occurs before time *simTime* we check the status of the entire service by analyzing the minimal path decomposition. We record the uptimes and the downtimes of the system during each *simTime* period (Figure 2.4). Critical component connections or disconnections cause the system status to oscillate from available to unavailable. The availability for that period can then be calculated as in Equation 2.2.

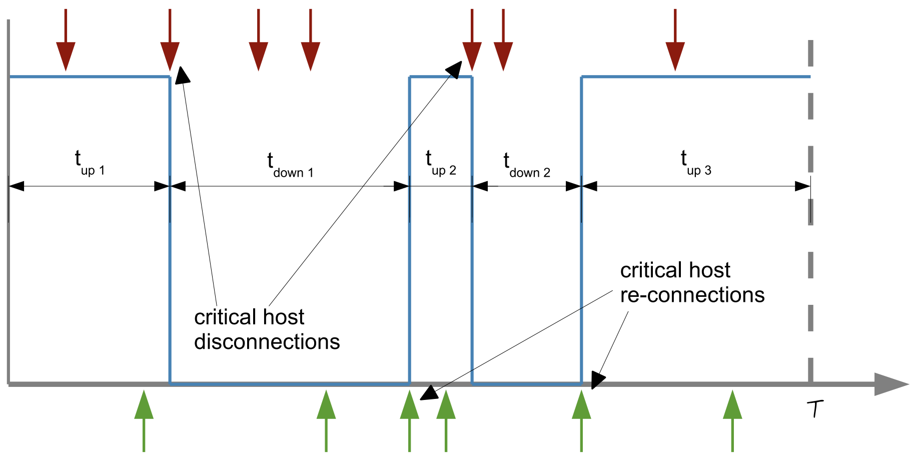

Figure 2.4: Discrete Event Simulation to reproduce service life-cycles

$$A_{sys} = \frac{\sum t_{up}}{simTime} \tag{2.2}$$

We can then repeat this process a large number of times (e.g., several hundred or even thousand times) to estimate the expected availability of the service. Additionally, as described in [89], it is easy to compute a confidence interval for the expected availability, so that we can have a measure of the estimation accuracy.

2.4.2 Pseudo-code of our hybrid approach

In this section we present a detailed pseudo-code version of our hybrid approach, introduced at the beginning of this section. For readability issues, we have divided the pseudo-code in two parts: (a) the node search algorithm (Algorithm 1); and (b) the system availability simulation algorithm (Algorithm 2). As mentioned earlier, the latter is used several times throughout the main procedure to estimate the system availability whenever a new configuration is proposed.

Remark that the same heuristic framework could be used with other methods to estimate the system availability. Similarly, a different search heuristic framework could be employed instead of the one presented here. This modular design opens future research lines to explore.

The Algorithm 1 requires the following inputs: (a) information about the nodes or community resources, including their individual costs and their individual availability levels;

(b) the logical topology of the service, which defines the operational conditions to be satisfied (E in Equation 2.1);

(c) the time $runTime$ the algorithm will run;

(d) the time period $simTime$ in which resources will be assigned to the service;

(e) the user's threshold for the service availability, i.e., the minimum availability level that the service must satisfy.

At the beginning, all the available nodes are sorted by descending availability level and an initial (dummy) solution is constructed by selecting the *best* free nodes, i.e., those resources offering the highest availability levels. This is the simplest *greedy* approach, but yet is the method used currently in the targeted systems. The system availability for this initial solution is computed using the system availability simulation algorithm in Algorithm 2 for 30 complete life-cycles. Notice in the implementation of Algorithm 1 we fixed the number of iterations every time Algorithm 2 is also called to 30. This number provides enough significance to the simulation results as it restricts the computing time when considering larger instances.

Then, an iterative process is started to seek for cheapest service deployments that fulfill the availability restriction. We do so by switching a number N of nodes from the current service deployment by N nodes from the idle ones (`shiftNodes` method). The N nodes and their replacements are selected randomly from both lists (as in the example of Figure 2.5) and we keep the cheapest solution as the best solution found by our method. Alternatively, in order to let the algorithm escape local optima, some iterations may accept a more expensive solutions (but still satisfying the availability restriction) as a valid solution (lines 19 to 21 in Algorithm 1). This feature and the random host shifting makes our method able to explore a wider set of the solution space of service deployments and hence obtain better results.

After the given $runTime$, the algorithm returns the best solution found so far: the one that fulfills the service availability requirement at the lowest cost observed.

Algorithm 2 presents our system availability simulation procedure. A number of iterations

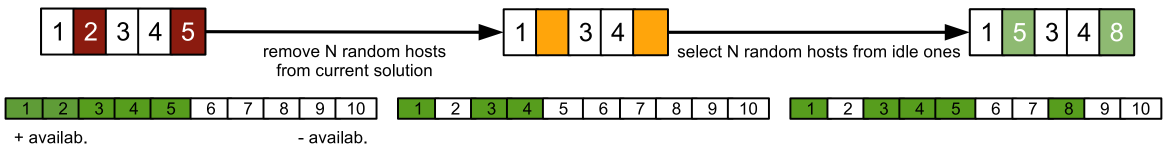
1 2 3 4 5
remove N random hosts from current solution
1 3 4
select N random hosts from idle ones
1 5 3 4 8
1 2 3 4 5 6 7 8 9 10
+ availab.
- availab.

Figure 2.5: The `shiftNodes` method replaces hosts in the current solution by idle nodes in the system

are run and, at each iteration, a complete life-cycle of the system is randomly generated. This life-cycle elapses from time 0 to the life-time value *simTime* defined by the user. This life-cycle is generated using discrete-event simulation with two types of events: node failures (disconnections) and node repairs (recoveries), as explained above.

Each time a node failure or repair event occurs, the system status is sensitive to change. Therefore, the system availability at that instance is checked. If it happens to be operative, the time until the next event is accounted as uptime. For the next event, the algorithm repeats the same procedure, until the event time reaches the *simTime* value. The service availability is then estimated as in Equation 2.2. Iterating this process a significant number of times (*maxIter* in Algorithm 2) produces a fair estimate value of the system availability. Of course, the accuracy of the estimate is directly related to the number of iterations.

Nevertheless, running more iterations requires more computation time, so a balance must be found for the maximum number of iterations to run. In general, relatively short simulations including just a few hundred iterations should be enough to get a *good* estimate of the system availability level. If necessary, however, longer simulations including thousands of iterations can be employed with the final solution provided by our methodology in order to give a more accurate estimate for that best-found solution.

2.5 Methodology evaluation

We split the validation of the proposed methodology in two different cases, depending on the scale of the system. In this scenario, we refer to the scale of the system as the number of nodes participating in the platform.

First, we validate the methodology previously proposed by comparing the solutions it

Algorithm 1 Replica-to-Host assignment search algorithm

```
Require: $nodes, topology, runTime, simTime, availabThreshold$
1: $nodes \leftarrow sortByAvailability(nodes)$
2: $baseSol \leftarrow buildGreedySol(nodes, topology)$
3: $bestSol \leftarrow baseSol$
4: $delta \leftarrow 0$
5: $lastImprovement \leftarrow 0$
6: $lastMovementWasAnImprovement \leftarrow false$
7: $runTime \leftarrow currentTime + runTime$
8: while $currentTime < runTime$ do
9:     $newSol \leftarrow shiftNodes(baseSol, nodes)$
10:    if $availab\_sim(newSol, topology, simTime, 30) > availabThreshold$ then
11:        $delta \leftarrow cost(newSol) - cost(baseSol)$
12:        if $delta < 0$ then
13:            $lastMovementWasAnImprovement \leftarrow true$
14:            $lastImprovement \leftarrow -delta$
15:            $baseSol \leftarrow newSol$
16:            if $cost(baseSol) < cost(bestSol)$ then
17:                $bestSol \leftarrow baseSol$
18:            end if
19:        else if $delta > 0$ and $lastMovementWasAnImprovement$ is $true$ and $delta <=$
           $lastImprovement$ then
20:            $lastMovementWasAnImprovement \leftarrow false$
21:            $baseSol \leftarrow newSolution$
22:        end if
23:    end if
24: end while
25: return $bestSol$
```

provides with the optimal solutions obtained for a set of small-scale instances in which several simplifying assumptions were made. Then, we considered large-scale instances, in which we compared the solutions generated by our approach with those obtained using *greedy* rules typically employed in real-life Internet computing systems, a Multi-Start approach and a Simulated Annealing algorithm used in a similar situation.

In order to complete these numerical experiments, the simheuristic was implemented as a Java application. All the experiments in this work were executed on a desktop computer with an Intel Core i5-2400 processor and 4 GBytes of RAM memory running Ubuntu Linux 13.04 and the Oracle Java Virtual Machine 7-64bits.

2.5.1 Optimal values in small-scale scenarios

Figure 2.1 showed earlier a graph representation of a distributed service which requires seven nodes to operate. Notice that the service can be seen as a system of interrelated tasks that

Algorithm 2 availab_sim(): System Availability Simulation

```
Require: $solution, topology, simTime, maxIter$
 1: $nIter \leftarrow 0$
 2: $A_{sys} \leftarrow 0$
 3: while $nIter < maxIter$ do
 4:     for all $node$ in $solution$ do
 5:         $dist \leftarrow getFailureDistribution(node)$
 6:         $eTime \leftarrow generateRandomTime(dist)$
 7:         $addFailure(eventsList, node, eTime)$
 8:     end for
 9:     $systemStatus \leftarrow true$
10:     $t_{up} \leftarrow 0$
11:     $pastTime \leftarrow 0$
12:     $currentTime \leftarrow 0$
13:     while $currentTime < simTime$ do
14:         $nextEvent \leftarrow nextEvent(eventsList)$
15:         $currentTime \leftarrow min(getTime(nextEvent), simTime)$
16:         $node \leftarrow getNode(nextEvent)$
17:         if $systemStatus$ is $true$ then
18:             $t_{up} \leftarrow t_{up} + (currentTime - pastTime)$
19:         end if
20:         if $nextEvent$ is a failure then
21:             $dist \leftarrow getRepairDistribution(node)$
22:             $eTime \leftarrow generateRandomTime(dist)$
23:             $addRepair(eventsList, node, eTime)$
24:         else if $nextEvent$ is a repair then
25:             $dist \leftarrow getFailureDistribution(node)$
26:             $eTime \leftarrow generateRandomTime(dist)$
27:             $addFailure(eventsList, node, eTime)$
28:         end if
29:         $systemStatus \leftarrow checkSystemStatus(topology, nextEvent)$
30:         $pastTime \leftarrow currentTime$
31:     end while
32:     $A_{sys} \leftarrow A_{sys} + \frac{t_{up}}{simTime}$
33:     $nIter \leftarrow nIter + 1$
34: end while
35: return $\frac{A_{sys}}{nIter}$
```

must be completed, over a time period, by this set of resources. The service will be available for the users if some logical conditions are satisfied. Thus, at any time there must be some nodes offering each of the service core tasks. The logical conditions define an underlying logical topology for the service to be operative. This topology is represented by the oriented edge connections in the directed graph.

Even though the existence of host availability behavior traces from a plethora of Internet distributed computing systems (like the ones collected in the Failure Trace Archive [76]), using them would require to build models for failure and repair distributions describing every host availability intervals. Since we wanted to asses the operation of our methodology rather than a model, we chose to use artificially-generated distributions. Thus, we isolated the operation of our methodology and proved the goodness of our proposal.

In this example, we will assume that the user specifies a minimum availability level of 90% of the time for the service during a period of 0.4 time units. Also, we will assume that for each resource, both its cost as well as the probability distribution modeling its failure and repair times are given as inputs. These inputs are shown in Table 2.1, where the mean availability of each node is computed as the quotient between: (a) the Mean Time To Failure (MTTF), and (b) the sum between the MTTF and the Mean Time To Repair (MTTR).

Values in Table 2.1 were artificially generated in such a way we obtained a set of nodes with mean availability levels that made interesting for contributory computing. Also, the cost associated to each node is correlated to some extent with node's mean availability, so the most available nodes are more likely to be also the most expensive ones.

Another simplifying assumption in this example is that only failure times are considered —i.e., no repairs are allowed. This assumption reduces the computing time needed to obtain the service availability for each new configuration generated during the search process and ease to shorten the exhaustive search while keeping our methodology as it was designed.

Our optimization problem consists then in selecting the set of seven ordered nodes, among the available ones, which minimizes the service cost (sum of nodes' costs) while providing a service with an availability level above the user-defined threshold.

Table 2.2 shows the results obtained for different instances of this problem. These instances differ in the number of available nodes in the community. The optimal value has been obtained using exhaustive search. Computational times employed by exhaustive search

Table 2.1: Probability distributions and costs associated with each host

Node ID	**Failure Distribution**			**Repair Distribution**			**Mean availab.**	**Cost**
	Type	**Shape**	**Scale**	**Type**	**Shape**	**Scale**	$\frac{MTTF}{MTTF+MTTR}$	
Node 1	Weibull	1.8	2.8	Weibull	1.3	0.3	0.9	18.96
Node 2		1.7	2.7		1	0.25	0.91	16.82
Node 3		1.6	2.6		1.5	0.4	0.87	17.83
Node 4		1.6	2.5		0.9	0.35	0.86	13.1
Node 5		1.4	2.4		0.8	0.25	0.89	17.21
Node 6		1.2	2.2		1.2	0.35	0.86	15.74
Node 7		1.3	2.3		0.95	0.4	0.84	16.41
Node 8		1.1	2.5		0.8	0.22	0.91	9.38
Node 9		1.4	2.8		0.6	0.36	0.82	14.78
Node 10		1.7	2.1		0.95	0.35	0.84	13.63
Node 11		1.6	2.7		0.78	0.26	0.89	15.72
Node 12		1.1	2.4		0.73	0.32	0.86	10.54
Node 13		0.9	2.9		0.87	0.29	0.91	18.71
Node 14		1.9	2.7		1.75	0.34	0.89	13.77
Node 15		0.8	3.1		0.93	0.31	0.92	14.58
Node 16		1.3	2.6		0.5	0.1	0.92	10.08
Node 17		0.75	1.8		1.2	0.28	0.89	14.59
Node 18		1.6	2.3		1.6	0.18	0.93	11.29
Node 19		1.1	2.9		0.8	0.26	0.9	9.93
Node 20		1.6	2.2		1.4	0.32	0.87	15.67
Node 21		0.8	2.9		0.75	0.1	0.97	12.1
Node 22		1.3	2		1.3	0.24	0.89	17.7
Node 23		1.8	2.3		0.92	0.36	0.85	11.36
Node 24		0.95	3.2		0.68	0.15	0.94	14.27
Node 25		1.5	2.8		1.4	0.42	0.87	15.56
Node 26		0.83	2.1		1.2	0.6	0.8	9.3
Node 27		1.2	3.2		0.6	0.3	0.87	12.06
Node 28		1.1	1.98		1	0.2	0.9	12.5
Node 29		0.79	2.3		0.8	0.34	0.87	17.2
Node 30		1.6	2.6		1.2	0.16	0.94	11.23

are given just as a reference, since we were only interested in the availability and cost values of the optimal solutions.

From the results in Table 2.2, the following observations can be made: (a) our approach is able to find an optimal solution in all the small-scale instances considered; (b) in all these instances, the computational times needed by our approach to find an optimal solution are below one second, while an exhaustive search method might need several hours to find an optimal solution due to the combinatorial explosion nature of this problem (Figure 2.6); and (c) from Figure 2.6, it can be noticed that the computing time employed by our approach

Table 2.2: Comparison between optimal solutions and our solutions

# hosts	**Optimal Solution**			**Our Best Solution**		
	Cost	**Availab.**	**Time** (s)	**Cost**	**Availab.**	**Time** (s)
10	101.28	0.91	5.073	101.28	0.91	0.038
15	89.78	0.91	229.232	89.78	0.91	0.084
20	78.09	0.91	2,620.736	78.09	0.91	0.133
25	74.36	0.91	15,814.235	74.36	0.91	0.167
30	73.29	0.90	63,461.778	73.29	0.90	0.268

—in solving these small-scale instances— is significantly shorter than the exhaustive search approach (few orders of magnitude lower).

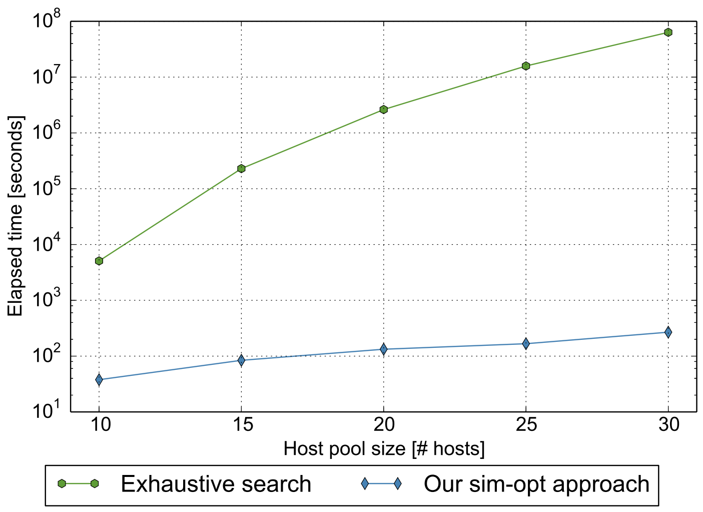

Figure 2.6: Computational times employed in obtaining the optimal solution. Remark the logarithmic scale on the vertical axis.

2.5.2 Performance evaluation in large-scale scenarios

We are interested now in analyzing how our approach performs in larger scenarios. These scenarios are characterized by large-scale distributed communities composed of hundreds or even hundreds of thousands of hosts with random failure times as well as with random repair times.

As discussed earlier, due to the complexity of the associated stochastic optimization problem, in these cases it is not possible to guarantee optimality of the solution in reasonable time periods. Therefore, we will not be able to compare the service deployment costs provided by our solution against the optimal cost. However, we can compare our solutions against some other well-known algorithms or proposals in the same direction: (a) the *greedy* solutions which are employed in Internet computing for solving similar situations (such as CoDeS [27] or LaCOLLA [90]) (b) a fast but simple *Multi-Start* search algorithm; and (c) the *Simulated Annealing* (SA) approach by Attiya et al. [79], which we implemented using the parameters fixed in the original paper. In all cases, the search method was combined with our availability simulator to determine the reliability degree of each of the generated service deployments. Likewise, we can also analyze the computing times based on our approach as well as other approaches in finding *good* solutions to these realistic scenarios.

We will use in our tests the same topology and desirable service availability level as in the previous experiment (also from Figure 2.1). The simulation time horizon is now fixed to 0.9 time units, so the effects of repair events can be perceived. Though, the list of available resources to select from (all the hosts in the system) will be much larger in this case. In particular, we will consider four different scenarios, with 500, 1,000, 10,000 and 100,000 hosting nodes. In order to easily generate the input data for such a large number of resources, we have used the rules detailed in Table 2.3 to define the properties of each node. All tests were run for a maximum time of 300 seconds.

Table 2.3: Rules for generating input data for each node

Node ID		Node N
Failure Distribution	**Type**	Weibull
	Shape	`Uniform(0.8, 2)`
	Scale	`Uniform(0.5, 4)`
Repair Distribution	**Type**	Weibull
	Shape	`Uniform(0.5, 1.75)`
	Scale	`Uniform(0.2, 1.8)` $\frac{MTTF}{MTTF+MTTR}$
Cost		$10 \times (availability +$ `Uniform(0,1)`$)$

Again, values in Table 2.3 were artificially generated in such a way we obtained a set of nodes with mean availability levels that made interesting for contributory computing. The cost associated to each node is also correlated with node's mean availability. From the host availability information in Table 2.3, we obtained four different sets of nodes, with the main host availability of each as shown in the histograms in Figure 2.7.

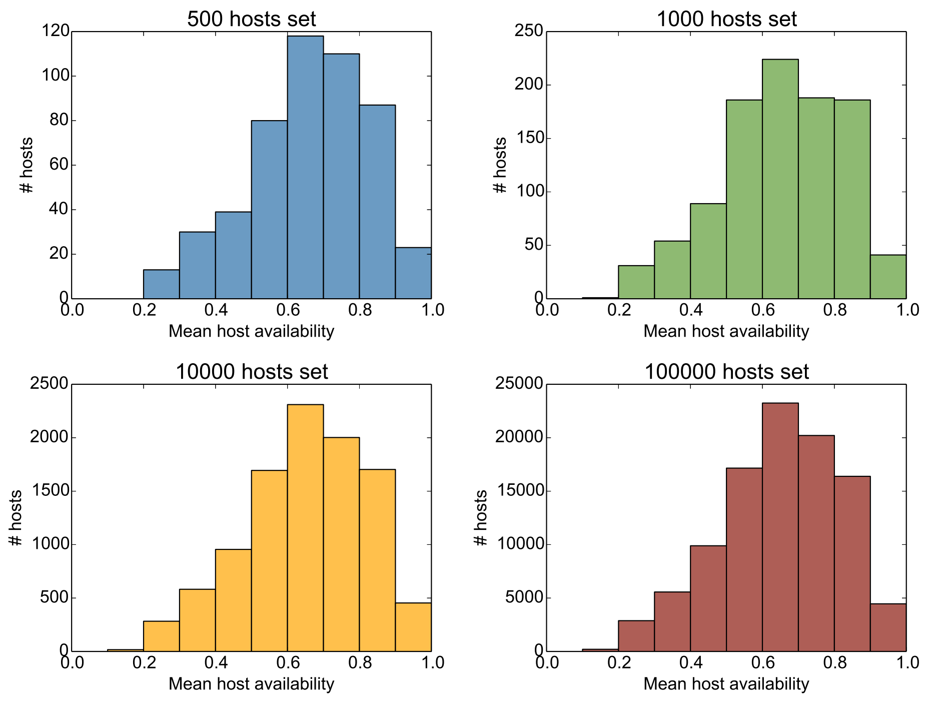

Figure 2.7: Host mean availability for the four node sets used in this experiment

Table 2.4 shows the results obtained using our simulation-optimization approach and how they compare against the ones obtained using the *greedy* rule of selecting the nodes with the highest availability level among all the nodes in the list.

From data in Table 2.4, the following observations can be made: (a) in all cases, our approach has been able to noticeably reduce the costs of the service deployment provided by other methods, with an average cost reduction of 58.3% from the *greedy* approach; (b) despite the size of the list of resources, our approach is able to provide *good* solutions —with important reduction of deployment costs— in *reasonable* computing times (under 300 seconds).

We recognize the performance of the SA approach implemented for this experiment could be further improved by fine-tuning the internal parameters of the algorithm (initial temperature, cooling factor and increasing factor). Thus, we believe that would lead the SA algorithm to obtain results of similar quality as ours. However, this fact also shows the goodness of our approach, since it requires less input parameters and, therefore, simpler fine-tuning processes. This trait makes it very interesting for heterogeneous scenarios like contributory communi-

Table 2.4: Comparison between *greedy* selection, Multiple Start algorithm, Simulated Annealing algorithm and our approach in large-scale scenarios (max. time = 300 seconds)

# hosts		500	1,000	10,000	100,000	**Average**
Greedy	**Cost**	74.53	72.26	79.72	102.71	
	Availability	0.97	0.98	0.98	0.98	
Multiple-Start	**Cost**	53.94	64.03	53.59	54.01	
	Availability	0.90	0.90	0.90	0.91	
	Gap greedy	27.63%	11.39%	32.78%	47.42%	29.8%
Sim. Annealing	**Cost**	42.08	43.01	47.49	55.6	
	Availability	0.92	0.90	0.90	0.92	
	Gap greedy	43.54%	40.48%	40.43%	45.87%	42.6%
Our Best	**Cost**	33.91	32.41	31.88	37.41	
	Availability	0.91	0.90	0.92	0.93	
	Gap greedy	54.5%	55.2%	60.0%	63.6%	58.33%

ties, since the methodology behaves properly in different circumstances with neither changes nor fine-tuning processes.

Figure 2.8 shows, for each of the considered instances, how the solutions provided by all the compared methods evolve —in terms of costs— with computing time.

Notice that the solutions obtained in just a few seconds are already far cheaper (almost a 50% cost reduction after only 10 seconds of execution) than those provided by the *greedy* approach —i.e. our approach can provide *good* solutions in real-time. It is also remarkable that our approach provides always cheaper service deployments than any of the compared methods, even when longer execution times are allowed.

2.6 Our simheuristic applied in a distributed social network use case

In the recent year, our research group proposed several distributed systems based on non-dedicated resources. One of these proposals is HorNet[1] [P1, P9], a microblogging service for *contributory social networks* which aims at overcoming dependency from current centralized microblogging services while making use of existing underutilized resources.

[1]`http://dpcs.uoc.edu/joomla/index.php/software/hornet`

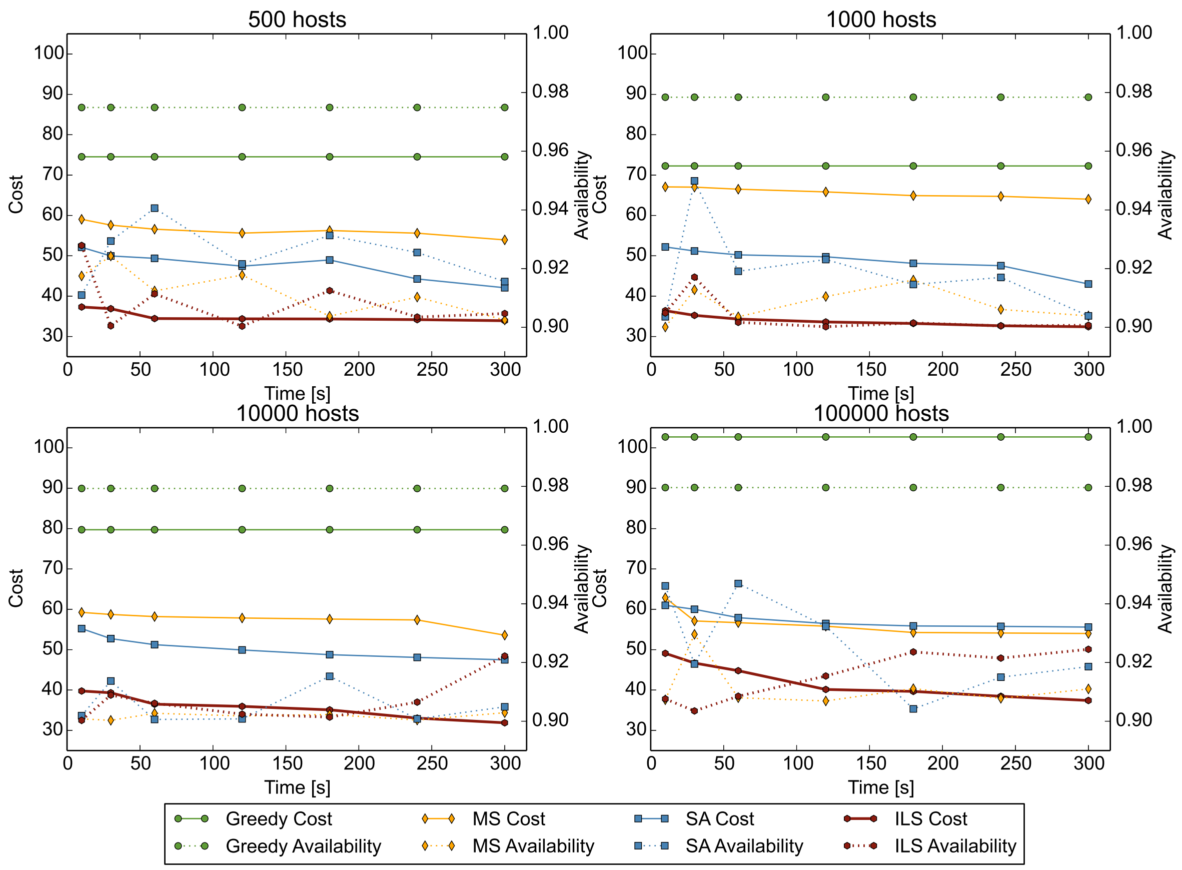

Figure 2.8: Comparison between the solution obtained by the four tested procedures in each of the studied host sets

We briefly discuss the need for a decentralized microblogging service and its feasibility in this section. In addition, present a use case of this application in which we apply our availability-aware host selection simheuristic to select the hosts to support the service replicas.

2.6.1 Distributed or decentralized social networks

Online social networks such as Facebook, Twitter or LinkedIn[2] play an important role in everyday life for a large amount of people. They have reached hundreds of millions of active users. Twitter is an example of a *microblogging* social network, which allows users to publish short messages (up to 140 characters in Twitter, called *tweets*) as well as *follow* other users and receive their messages. Twitter has gained a great social importance, and has been used,

[2]`http://www.linkedin.com/`

together with other social networking services, for public relations, marketing and networking by many businesses, as well as in organization of collective actions or to spread information about important news by and to the general public. However, the social importance of these services makes it paramount that they remain free and under control of the users.

Nevertheless, all major social networking platforms depend on a company which provides both the software and the resources where it is executed, and stores all the information and content generated by its users inside the social network. This allows the service provider, for instance, to control or even censor information about certain topics. Even if the service provider is trusted, a centralized service can be banned by governmental powers or the same provider, as has happened to Twitter [91]. Moreover, centralized services can collect, analyze, and correlate users' data, thus generating even more information about the identity and interests of a user than that included in his messages. This poses a critical privacy risk since this information can be compromised or revealed to interested third parties. At the same time, the contributory computing paradigm showed to be a valuable tool to utilize underutilized or surplus user-contributed computers as a service platform.

In a contributory community, HorNet components are deployed dynamically and users are free to decide when and which resources to contribute, keeping their freedom and ownership of their computers while accessing and supporting a service which is collectively maintained and owned. This collective ownership of the service, which is true in a physical sense (the service is running on machines owned by different people) prevents the service from being centrally controlled, censored or turned off (either by technical failures or by conscious will). In any case, a minimal user commitment is required to keep a minimal amount of resources contributed. Provided that these resources are contributed (by any subset of community members), any resource might enter or leave the community at will without compromising its regular operation. Otherwise, the community would not be able to support the intended services with the desired quality. This minimal amount of resources will depend on the size and the activity level of the community members.

Related work

Decentralizing social networks is not a new idea. Different solutions have been proposed to avoid depending on a centralized entity, mainly for privacy reasons [92]. In many of these works [93, 94, 95, 96], each user stores his own data in his own computer, in a trusted computer or in a paid cloud utility computing, retaining full rights to the content and sharing them with other users in a peer-to-peer basis. Availability of data depends on the availability of the computers hosting the data. Our proposal focuses on guaranteeing the availability of data regardless of the availability of each members' computer by deploying small services (that manage user's data) and a storage service in resources freely contributed to the community by its members.

Other proposals have tried to leverage peer-to-peer networks for social networking by distributing data among peers [97], but they have focused on the challenges regarding data privacy [98]. In contrast, in HorNet the service is split into low-demanding operations that are executed on the individual resources contributed by end-users. Private information, like profile information, can be easily stored safely using cryptography and be only revealed to authorized people. This way, the sensible data available to a user cannot be used to infer the interests of other users beyond what is revealed in their messages.

2.6.2 The HorNet architecture

We designed HorNet as a three-tier application (as described earlier in Section 1.3) on top of the CoDeS middleware implementation [27], used as a distributed directory service. In our prototype implementation, we relied on a dedicated web-server for the presentation layer and on an external storage facility for the data layer.

The microblogging service implements the logic as a set of smaller and stateless services, deploying an instance of each of them for every user in the system. These services control the messages the user sends and receives, her followers and other information. They use the storage service to provide persistence to tweets and metadata and the middleware handle the resource aggregation and manage the service replica allocation and reallocation on host churn.

Scalability (in number of users) and service availability guided the design decisions of HorNet. Scalability is achieved by decomposing the microblogging functionality it provides in a large set of small low-requiring components, each of which deals with one of the functionalities of each user. Specifically, for each user U_i, the HorNet core deploys six components:

a *Messages*: messages published by user U$_i$. When the user publishes a new message, this service sends it to all her followers.

b *Timeline*: messages received by user U$_i$

c *Followers*: users that follow user U$_i$

d *Following*: users followed by user U$_i$

e *Mentions*: messages that have a mention to user U$_i$

f *Profile*: general information about user U$_i$, such as full name, avatar, etc.

When a user is created, a new instance of each of the six services is deployed. From that moment on, availability is provided by the directory service, freeing HorNet of the burden of dealing with the dynamic and heterogeneous nature of domestic computers.

Figure 2.9 presents a HorNet social network with many users and many contributed computers. From all these users, the figure focuses in: Alice, Bob, John and Mary. To simplify, the figure only includes the *Timeline* and *Messages* service of each of them. Alice, Bob and John are connected to HorNet using a HorNet client installed in their home computers. Mary is not currently connected. Finally, John, Bob and Mary follow Alice. Alice sends a tweet *(1)*. The tweet is first sent to her *Messages* service *(2)* that stores it persistently. From *Alice.Messages*, the tweet is sent to the *Timeline* service of her followers (Bob, John and Mary) *(3)* and her own *Timeline (3)*. Notice that even though Mary is not connected, her *Timeline* receives the tweet. All these *Timeline* services store it persistently in the storage layer. Alice, Bob and John's web containers perform periodical connections to their *Timeline* services *(4)* from which they refresh the received tweets.

In case of any service disruption, recovery mechanisms should restore a new replica. In this scenario, HorNet services should perform regular checks for new updates with other components, in such a way lost or not received messages could be eventually acquired.

2.6.3 Feasibility evaluation

According to our simulations using the HorNet implementation described above, the 99th percentile of the time it takes for a message sent by a user to reach the *Timeline* service of

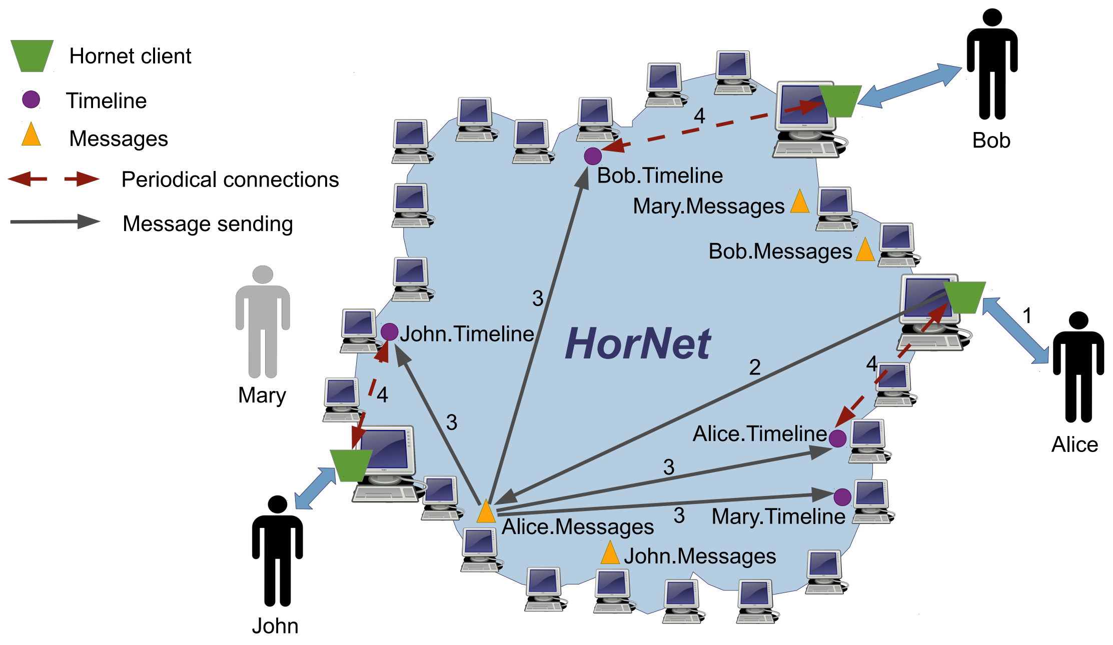

Figure 2.9: HorNet social network. To simplify, the figure only includes the *Timeline* and *Messages* service of users following Alice.

a follower is 3.66 seconds. Furthermore, the time it takes for a user to retrieve 20 received tweets is 1.9 seconds (99th percentile), which is reasonable from a user point of view. Our current design retrieves last 20 tweets from the user *Timeline* when the user connects and in blocks of 20 tweets each time she asks for older messages. However, we still lack a real user community which would allow us to perform actual performance and user satisfaction measurements.

In order to asses the feasibility of such a system in a real-world community, we made an estimation of system-wide bandwidth usage and the amount of data generated based on the typical user behavior found in Twitter. We consider bandwidth and disk space would be the limiting factors in a real deployment, since the computational power and memory required by each HorNet service is very low.

Usage data estimation

We used Twitter usage data summarized in a report by Barracuda Labs [99] to build a simple user activity model. On the one hand, 79.2% of Twitter users send less than 1 message daily and 95% of them send less than 10. On the other hand, 83% of Twitter users follow less than 100 friends and are followed by less than 100 of them.

Thus, we modeled a typical Twitter user as follows: she would send 10 messages every day, would follow 100 friends and would be followed by 100 other users. This makes an overestimation of the actual average values found in Twitter. We also made the assumption that each tweet would be represented by 200 bytes, containing the message itself and all the needed meta-data (i.e. author, timestamps).

Bandwidth and Storage

Taking into account our basic user model and the data replication strategy presented by Bhagwan et al. [71], which computes the required replication to achieve a specific target data availability, we estimated the storage and bandwidth usage of our system.

Let us consider $\mathtt{D}_{tweet}$ as the volume of data a user message generates (calculated as shown in Equation 2.3a), $\mathtt{R}$ as the number of replicas needed to guarantee data availability (determined as detailed in Equation 2.3b), $\mathtt{a}_{sys}$ as the overall required system availability, $\mathtt{a}_{host}$ as the mean host availability and $\mathtt{N}_{tweets}$ as the number of daily messages sent by a single user. We can determine the storage volume each user has to contribute by counting the number of messages she sends and receives. If we assume that each user contributes a machine, with an average availability a_{host}, then it is easy to estimate the average number of users each host has to manage. Adding the fact that information is stored replicated, we can compute the amount of storage that each user has to contribute daily with the expression shown in Equation 2.3c and the required bandwidth per user dividing this amount of data by the length of a day.

$$D_{tweet} = (1 + N_{followers}) \times L_{tweet} \times R \tag{2.3a}$$

$$R = \frac{log(1 - a_{sys})}{log(1 - a_{host})} \tag{2.3b}$$

$$D_{host} = \frac{D_{tweet} \times N_{tweets}}{a_{host}} \tag{2.3c}$$

Providing a highly available service is one of the challenges of our proposal. Therefore, we fixed a four nines (available 99.99% of the time) as the target availability for our system. We also assumed that sent and received messages are stored for one month. Users could choose to store older important messages in their local computer. Figure 2.10 shows the estimated required bandwidth and disk space from each node in function of average host availability. The required bandwidth in the upstream and the downstream channels are the same.

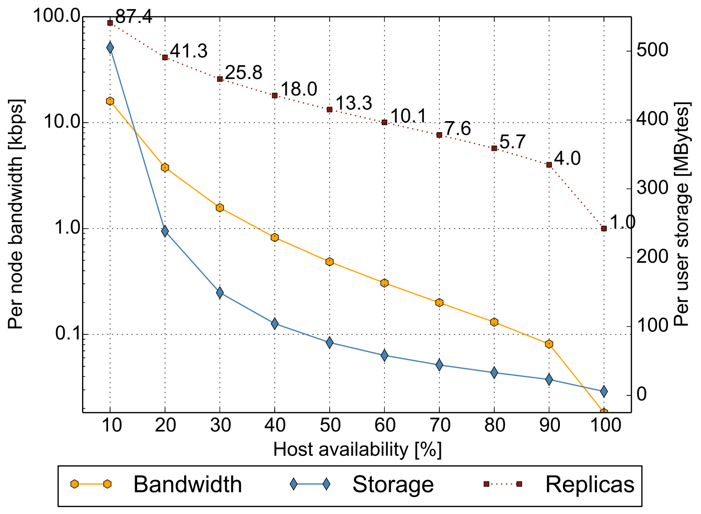

Figure 2.10: Required bandwidth and storage capacities from each node and replicas per service in HorNet

Figure 2.10 shows that, for example, in the case of an average host availability of 60% each host should contribute around 58MB of disk space and a constant bandwidth of around 0.31 kbps. For 80% and 40% availability storage is 33MB and 104MB respectively and bandwidth 0.84kbps and 0.13kbps. This states that such a kind of service deployment would be completely feasible with current domestic computers storage volumes and the Internet connections available in most parts of the world.

High demanding scenario

To reinforce our position on HorNet's feasibility, we show an extreme example of service usage from a high demanding user who could potentially be interested in this contributory social network. Consider the case of an organization like Greenpeace, the environmental NGO, which could be interested on using the resources contributed by its affiliates[3] to freely spread their messages. This way, they would have no dependence on third-party organizations.

From the usage statistics of the organization's Twitter account (285,000 followers and 6 tweets/day), we estimated that Greenpeace's *Messages* service would have to send 6 messages per day to 285,000 users (replication would be managed by the receiving *Timeline* services, which would store the tweet in the required number of replicas). Therefore, the bandwidth required by this *Messages* service would be higher than the average. The node responsible for dealing with Greenpeace tweets (the node hosting the *Messages* service) should therefore be offering a constant upload rate of approximately 30.9 kbps. If this node had a typical upload bandwidth of 300 kbps, the time needed to send a message to all the followers would be around 25 minutes. This delay could be tolerable, since delay is not a key feature in the original Twitter service, which offers eventual consistency due to its database schema over Cassandra [100]. Moreover, the task of delivering tweets from high demanding users could be split among several hosts in order to reduce the required bandwidth from a single desktop computer.

2.6.4 Availability-aware allocation in HorNet

This section presents the simulation of a contributory community where a basic social network is deployed. We took as reference the six atomic components for each user in HorNet, and we introduced redundancy in the two key tasks for the system. An instance of each service needs to be available at any time in order to consider that the social network is fully operative. Therefore, we modeled the system as shown in Figure 2.11, on which Services `Messages` and `Timeline` are supported by two instances while the rest (`Followers`, `Following`, `Mentions` and `Profile`) are only instantiated once.

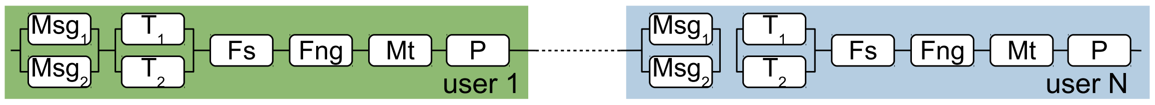

Figure 2.11: System model for a N-users HorNet deployment

[3]Greenpeace states they have around 2.8 million economical supporters worldwide and their staff is around 2,400 people worldwide (July 2011).

Let us imagine a college decides to offer a community-based microblogging service to their students as a use case example. Assume the community is supported by 100 well-provisioned computers donated by the institution. Besides of this, 500 personal computers are contributed by the enrolled students. Assume the availability and cost information of the first 500 nodes in the previous example in Section 2.5.2 and the 100 stable node information is generated as in Table 2.5.

Table 2.5: Rules for generating input data for each stable node

Node ID		Stable Node N
Failure Distribution	**Type**	Exponential
	Rate	`Uniform(3, 7)`
Repair Distribution	**Type**	Exponential
	Rate	0.05
Availability		$\frac{MTTF}{MTTF+MTTR}$
Cost		$10 \times (availability + \texttt{Uniform(0,1)}) - 5$

Parameters in Table 2.5 were selected to model highly reliable hosts, with long enough available intervals and very quick restores from failures. We keep the same cost-availability relationship than with other hosts, so this stable nodes should be slightly more expensive than user-contributed resources.

We plot in Figure 2.12 the mean availability histogram for the hosts considered in this new scenario. As expected, the addition of 100 institutional hosts with high mean availability increases the mean availability of the nodes in the system. Combining non-dedicated and dedicated hosts, more reliable service deployments could be achieved.

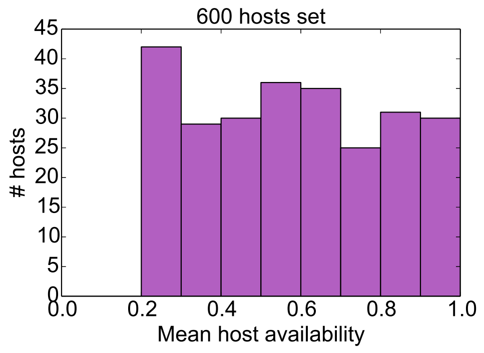

Figure 2.12: Host mean availability for the node set used in this experiment

In this scenario, three students decide to deploy their microblogging service to communicate privately while working on an assignment. The system topology is composed then by 24 components disposed as in Figure 2.11 and the students only require an 80% of service availability.

We performed the same experiment than in Section 2.5.2: we sampled the cost and the availability of the best found solutions after several execution times (as plotted in Figure 2.13). In this case, we only compared our solution to the *greedy* deployment, as this is the only method currently implemented in similar systems [27, 90].

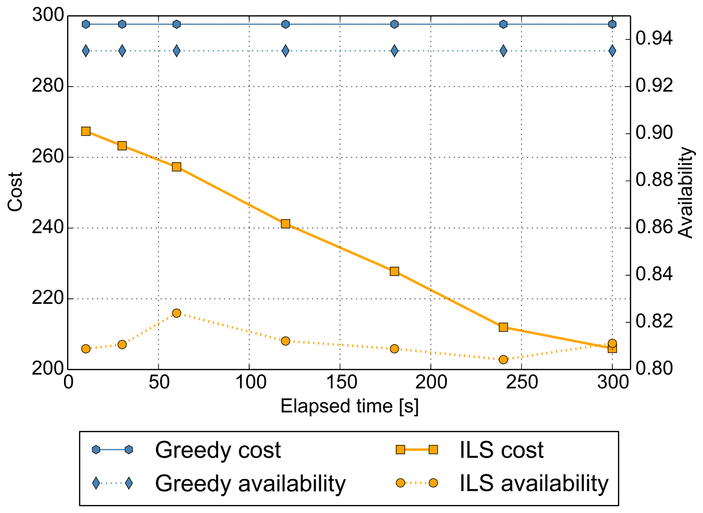

Figure 2.13: 3-user HorNet deployment in a University-aided community

Devoting computational efforts at the host selection stage could result in significant cost savings. As can be observed in Figure 2.13, the larger the time spent on scavenging the node set, the lowest resulting costs from service deployments, while the service availability level is degraded at a much lower rate.

This example showed our methodology is ready to be adapted to a real-world service deployment. The lack of an actual deployment of this system with real users prevented a test in real conditions. The results shown in Figure 2.13 clearly state our method could be embedded in a user-interactive service deployment procedure, as close to optimal deployments are found when only 10 seconds are given to run our host selection procedure.

2.7 Summary

The scale and relevance of existing distributed computing systems is rapidly growing with the increasing use of inexpensive commodity, heterogeneous, and non-dedicated resources. However, the narrow availability level of non-dedicated resources constitutes an important challenge for the range of possible applications of these systems.

To contribute solving this problem, this chapter presented an original simulation-optimization methodology for cost-effective and availability-aware service deployment. Our approach combines a heuristic framework —which iteratively constructs cost-efficient solutions for deploying services—, with a flexible simulation-based algorithm —which is used to ensure feasibility of each newly generated solution through estimation of the corresponding service-availability level. Therefore, our proposal can be used to promote efficient deployment of several Internet composite services in community environments. To the best of our knowledge and despite its relevance, this problem has not been solved before in this particular environment.

Thus, for example, previous proposals for availability-aware service deployment required the use of restrictive assumptions —e.g. identical replicas of a service, series-parallel topologies, small-scale scenarios, probability distributions employed, etc. According to the numerical experiments we performed, our approach is able to quickly provide optimal solutions in small-size scenarios, while it can be also used in more realistic scenarios to generate *good* solutions in real time, thus improving the greedy approaches typically used in Internet service deployment practices over contributory resources.

Our experiments proved our methodology to obtain good quality solutions in reasonable times. This would allow user-interactive service deployment procedures with close to optimal allocations. Moreover, improving the greedy solution also causes a proper resource usage. Thus, more services could be supported by the same number of resources or less resources could support the same number of services.

3 Issues of energy-aware allocation in distributed systems

This chapter discusses the energy matters of distributed computing platforms. We present two different strategies aimed to reduce the consumption of these systems, composed either by dedicated or non-dedicated computing resources.
The contents of this chapter were extracted from publications [P2, P4, P5, P11, P12].

3.1 Overview

Internet services have become extremely popular in current societies. End-users quickly adapted themselves to social networks of different nature, as well as to many different services including, among others web-based interactive email services, multi-purpose remote storage systems or user-owned on-line picture galleries. What all these services have in common is that they are supported by large —and often geographically distributed— computing infrastructures managed by remote companies. They are offered to the user as simple web pages or wizard-installed applications, needing nothing else but an Internet connection and a low-end device. As explained in Chapter 1, all the resources supporting these services are commonly known as a cloud platform. As the service popularity and demand grows at nearly-exponential rates, so do the datacenter infrastructures necessary to support these services.

Maintaining potentially large infrastructures needed by cloud computing is neither cheap nor environmentally friendly [101]. Most of them consume vast amounts of energy to run the servers, the network equipment and the cooling systems. Still, experts have estimated that only around 10% of the energy used in a datacenter goes to powering active servers. The rest of the energy is used for keeping idle servers waiting for activities from user requests. Governments and datacenter owners are starting to be concerned about the effects over the environment associated with the creation and maintenance of new cloud infrastructures.

So far, performance was the main criterion that guided the design, maintenance and operation of these computing centers. However, moving towards *green computing* has emerged as a tendency in most sectors and companies are now starting to consider their carbon footprint [102]. According to Lo and Qian, 'green computing refers to environmentally sustainable computing which studies and practices virtually all computing efficiently and effectively with little or no impact on the environment' [47].

Also, as stated by Laplante and Murugesan in [103], 'an increased awareness of the harmful effects of greenhouse gas emissions, new stringent environmental legislation, concerns about electronic waste disposal practices, and corporate image concerns are pushing business and individuals to go green' and 'business also need to look at green requirements from another viewpoint —that is, the implications of not going green in the context of stricter environmental regulations, stakeholder demands, competitiveness, branding and corporate image, and social responsibility'.

Therefore, it is necessary to take into account energy consumption, greenhouse gas emissions and the carbon footprint while optimizing costs and performance levels of distributed computing facilities, as clearly stated by Garg et al. [104]: 'there is an urgent need for energy-efficient solutions that can address the high increase in the energy consumption from the perspective of not only the Cloud provider, but also from the environment'.

Due to the large scale and geographical distribution of these facilities, as well as to the complexity of the services being offered, the resulting multi-objective optimization problem arises as an attractive challenge for the research community. The goal for current distributed computing platforms is to drastically reduce its environmental effects without affecting too much the service performance and quality. In this direction, Zhang et al. defined *green computing* as 'an emerging technology that applies intelligent optimization algorithms and advanced computing techniques to minimize energy consumption and reduce pollution on computing resources' [105].

In this chapter, we provide an overview of green computing issues related to optimization problems and we identify how classical techniques could be applied towards reducing the environmental effects of computing infrastructures. We review the current literature in the field of green computing, detail the detected problems on current Internet computing systems and show how similar problems were faced in the past. We classified the detected problems into those devoted to deal with dedicated infrastructures and those dedicated to manage non-dedicated systems.

In the case of dedicated infrastructures, there is a need to increase the number and size of datacenters offering to support all the more popular Internet services. However, this also implies a sharp annual increase in energy consumption, what made many research proposals to focus on different fields of green computing. Due to the large scale and geographical distribution of datacenter facilities, as well as to the complexity of the services being offered, the resulting multi-objective optimization problem arised as an attractive challenge. One of the most recurrent topic are scheduling algorithms, on which energy can be introduced as a variable.

In the second case, our vision is the contributory computing model favors the green computing postulates. Exploiting non-dedicated resources not only could avoid the need for large digital warehouses, but also optimizes the resource utilization ratio. We believe most of the user-contributed resources would be still consuming energy if they were not seized in a community platform, so gathering them for collective usage may result in lower energy consumption than hiring dedicated cloud platforms.

3.2 Green Internet Computing

Large datacenters supporting either grid or cloud computing aggregate a relevant number of dedicated servers in a relatively reduced space, what causes the heat density in these spaces to be high. Consequently, cooling systems become a must for the correct operation of these infrastructures. Though their consumption are usually underestimated, some studies claim they can make up an important percentage of the total energy consumption in the system. The Power usage effectiveness (PUE) is a measure of how efficiently a facility uses its power [106]. This parameter stands for how much of the energy is actually used by the computing equipment in contrast to cooling and other overheads. It is formally expressed as a ratio between the total facility power and all the equipment power. Accordingly to the results of a survey among large datacenter managers, current modern facilities offer an average PUE between 1.8 and 1.9 [107]. Whereas this number is clearly below the average 2.5 reported in 2011, it still means almost the same energy employed to power IT equipment is consumed by all the other accessory devices needed for the regular operation of a datacenter. Even so, this reduction shows the enterprises behind the clouds do actually matter about their energy consumption. Furthermore, all these facilities rely on backup protection mechanisms for the case of a power failure, either in the form of huge batteries or diesel generators. Hence, the carbon footprint of these facilities and its impact on the environment is large enough to consider taking measures to reduce it.

In parallel to the great success of cloud computing and the services it enables, desktop computers are continuously becoming more and more powerful. However, most of the end-users do not exploit the power of their own computers, due to the low-demanding tasks they perform locally, to the movement of some tasks to the cloud or because computers are left on when not being used. Volunteer computing proved to be a successful attempt of harnessing non-dedicated resources and some of the proposals for contributory computing presented in Section 1.3.1 pointed the energy consumption as a motivation [34].

Aggregating user-contributed resources is not only interesting by economic means but also because of the clouds environmental impact. While dedicated servers forming the traditional datacenters are located in the same physical space, donated computers are hosted and maintained by end-users at their homes or offices. Thus, the heat density in these cases is much lower thanks to the geographic disaggregation; the cooling system becomes no longer mandatory and there is a potential energy saving in this point.

However, some additional optimization issues must be addressed when employing community clouds. Probably the most noticeable of these issues is related with service reliability and availability, as previously discussed in Chapter 2. Replication strategies may cause the global system to be quite inefficient in terms of resource assignment and use. Also, distributing tasks behind services throughout different computers in Internet may require additional coordination procedures that might create some overheads both in terms of performance as well as in terms of complexity of the system. Still, adopting smart policies on replica placement and node selection procedures could also lead to a more efficient and environmentally friendly cloud model.

All in all, either if using a more decentralized architecture or not, in order to achieve the goal of an environmentally cleaner cloud, energy consumption must be considered in the design of the systems and services, the resource usage policies, the resource selection process and the scheduling of the supported services by the community cloud.

3.2.1 Optimization towards green computing

Issues regarding the development of environmentally sustainable computing have been discussed since the emergence of the clouds.

Duy et al. designed, implemented and evaluated a Green Scheduling Algorithm with a neural network predictor for optimizing power consumption in datacenters [108]. They predicted future load demands from historical data and turned off or restarted servers according to it. Through simulation, authors showed their method saved up to 46% of energy with insignificant demand drop. *Green* task-scheduling algorithms are presented in [105]. After a simulation experiment, the authors concluded that heuristically assigning tasks to computers with lower energy is significantly more energy-efficient than doing it randomly. Borgetto et al. studied the problem of energy-aware resource allocation for long-term services or on-demand computing tasks hosted in clusters [109]. They formalized the problem as NP-hard constrained optimization problems: (a) maximize job performance under energy consumption constraints; (b) minimize power consumption under performance constraints; and (c) optimize a linear combination of power consumption and job performance. They finally proposed several heuristics and used simulation to validate their approaches by comparing their results in realistic scenarios.

Chen et al. analyzed different algorithms for supporting connection-intensive Internet services [110]—-e.g. chat and video-conferencing services. These authors took into account several factors such as load distribution, user experience, and also energy consumption. Le et al. presented a framework for optimization-based request distribution among datacenters [111]. The novelty of their work consisted in that they introduced the energy consumption at the optimization level, seeking for datacenters in under-peak demands, datacenters close to a renewable energy source or in different time zones where the energy might be cheaper at that time. Their heuristic redirected user requests to those datacenters which could offer the required Service Level Agreement while minimizing energy consumption and operation cost.

Differences among the various data centers of the same company were considered in order to improve efficiency and reduce carbon emissions in [104]. These authors defined an energy model for data centers which included metrics to measure energy cost, carbon emission, profits, etc. On the basis of this model, they proposed heuristic scheduling strategies and evaluated their performance.

Careglio et al. analyzed energy-consumption implications from all the computer components [112]. Also, they presented a short overview on the advances of energy usage in network infrastructures. Moreover, in the context of large-scale distributed systems, these authors listed a set of best practices which could reduce energy consumption. These practices covered datacenter building-construction techniques, context aware buildings, as well as cooling systems.

Yuan et al. also reviewed different strategies for efficient energy utilization, ranging from server virtualization and consolidation to optimal operation of fans and other cooling equipment [113]. They discussed in more detail the case of cloud-based multimedia services, which posed specific challenges, such as larger storage and bandwidth requirements.

All the works cited above were mainly concerned with the energy consumed by the servers themselves. However, the fraction of the energy consumed by other devices, such as network infrastructure, is by no means negligible. Interest in this topic is more recent, albeit growing rapidly.

Berl et al. discussed such issues within the framework of the overall energy efficiency of different distributed information technologies, with emphasis in cloud computing [114].

GreenCloud is a simulation environment for energy-aware cloud computing datacenters [115]. Designed to forecast the energy consumption associated with each system component, it also tracks packet-level communication patterns and their energy usage. Authors simulated different datacenter schemes and demonstrated the utility of different power management procedures for either servers or network equipment, such as DVFS (Dynamic Voltage and Frequency Scaling) or DNS (Dynamic Shutdown). Lee and Zomaya discussed about energy wastage caused by under-utilized resources within a conventional cloud computing environment [116]. They argued that task consolidation is a good resource re-allocation procedure, thanks to the DVFS features included in modern processors. Two energy-conscious task consolidation heuristics were presented and analyzed, both of them reducing the energy consumption of clouds.

In another recent paper, Beloglazov et al. addressed the problem of efficient power management in data centers or processor farms [117]. They made a thorough review of previous work and proposed a technique for efficient power management based on virtualization. Their algorithms allocated virtual machines to actual servers dynamically, according to changes in the workload, and switched idle servers to sleep mode, thus minimizing energy consumption. Their assumption was that processors were the main power-consuming devices, which was not always the case in practice. Nevertheless, in contrast to previous approaches, their method was independent from the type of application or workload.

Nano Data Centers (NaDa) [118] were proposed as a new architecture that used ISP-controlled home gateways to provide computing and storage services. The challenge was to adopt an ISP-managed peer-to-peer model to form a fully distributed data center infrastructure to be exploited to offer Internet services to their customers. They built an energy saving model and stated their proposal could theoretically achieve up to a 60% of energy savings when comparing to traditional data centers. Their model was validated by studying a video-streaming service on different platforms and showed it could actually save around 20% of the energy that would be used by legacy data centers.

Schott and Emmen stated that the main advantage of distributed desktop-grids versus centralized-grid infrastructures was the minimal heat density of personal computers [119, 120]. Due to this fact, these computers did not require intensive air-conditioning systems, saving electrical power, money and reducing the pollution generated by datacenters. They proposed seven different methodologies as a collection of best practices, techniques and policies to reduce the environmental impact of desktop-grids, such as exploitation of natural ambient conditions, energy-profiling of running applications or reducing the necessity of air-condition.

Finally, it is worthy to mention that several researchers have agreed on identifying two main stages in the quest for sustainable information technologies [121, 122, 123]. According to these works, the first stage (*Green IT*) is mainly driven by the companies desire to reduce costs and increase profits. The second stage (*Sustainable IT*) is driven by broader concerns about the environment and rational utilization of resources. The aforementioned works discussed recent trends and business strategies regarding sustainable IT. In particular, Harmon and Demirkan reviewed the strategy followed by some important companies to cope with these challenges [122].

3.2.2 Open optimization challenges for green computing

From the analysis of the literature it is possible to conclude that there are several interacting factors involved in green computing optimization, e.g.: system architecture, resources geographic allocation, task-resource assignments, tasks scheduling, resource-operation policies, reliability of resources and Internet connections, availability of non-dedicated resources, energy prices in different countries, etc.

This rich environment offers an enormous variety of interrelated research optimization problems, many of which are multi-objective in nature, since system performance must be balanced with environmental sustainability.

We divided the open challenges in those devoted to systems composed by dedicated resources and those focused on systems formed by non-dedicated resources. This classification is not arbitrary, since systems with dedicated resources tend to aggregate similar and so homogeneous resources, while those including non-dedicated resources would show more dynamism and therefore will be more unpredictable.

3.3 Green computing challenges associated with dedicated resources

Most of the work dealing with green computing in the cloud focus on electricity consumption at server level within a single datacenter. The most common techniques are server virtualization and machine consolidation, DVFS or DNS.

As discussed earlier, several tasks-assignment and tasks-scheduling algorithms —usually heuristic in nature— have been already proposed. Most algorithms are non-preemptive, i.e. once a task is assigned to a server or scheduled in a given order, it cannot re-assigned or re-scheduled. In comparison, very few papers have addressed the scheduling problem in global terms, i.e. taking into consideration several datacenters at the same time. This global approach may lead to greater savings, and also poses interesting logistics-related challenges similar to those encountered in the context of logistics location/allocation problems or multi-depot routing problems [124]. Thus, for instance, Garg et al. divided the scheduling problem into two phases [104]: meta-scheduling or allocation/assignment —i.e. deciding the datacenter where the task will be executed—, and scheduling of tasks within the datacenter. In order to efficiently solve these challenges, heuristics and meta-heuristics algorithms can be employed either on their own or in combination with mathematical programming techniques.

Similarly, only a handful of papers deal with energy consumption of computing accessories —hard disks, network routers, etc. Moreover, to the best of our knowledge none of them integrates all devices together with the CPU. The amount of energy consumed by such devices is not negligible, and the need to manage them efficiently is becoming more and more evident in recent years. Here, methods from statistics and data mining can play a decisive role while identifying relevant factors, defining consumption indexes to be traced and developing regression (predictive) or forecasting models.

Another important aspect is the lack of methodologies able to support random behavior in real-life cloud systems —e.g. stochastic service demand, stochastic latency times, stochastic availability of resources, stochastic processing times, etc. Due to its importance in most real-life scenarios, most optimization methodologies in the green computing field should take into account the system random behavior as it already happens in similar problems [125]. Clearly, methodologies based on simulation, Markov chains [81] or dynamic programming can be necessary in order to include this random behavior of the system into realistic optimization models.

3.3.1 Task execution scheduling

One of the critical issues in distributed computing is the scheduling of tasks (jobs) to be processed by servers (machines) in large computing infrastructures. Optimal scheduling policies can represent noticeable reductions in the total time necessary to process a set of programmed tasks, also known as makespan (as explained graphically in Figure 3.1).

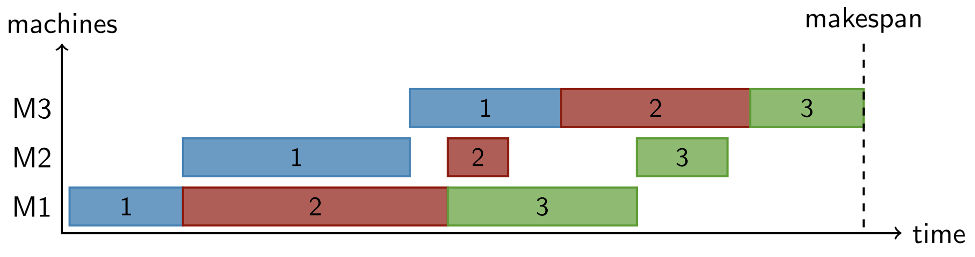

Figure 3.1: A simple flow-shop scheduling problem with three jobs and three machines

In turn, noticeable reductions in makespan can imply significant savings in energy consumption, since advanced techniques like DVFS, DNS or Virtual Machine (VM) consolidation can be exploited. Thus, computing infrastructures are utilized in a more efficient way and require less total power to complete the assigned tasks.

Despite the existence of a large number of scheduling-optimization algorithms, most of them assume deterministic values for the time each task requires in order to be processed by each server. However, this is a quite unrealistic assumption, since these task-server processing times are random variables —as opposed to deterministic values— in most realistic scenarios.

In order to contribute to fill this gap, this section discusses how Monte-Carlo simulation can be combined with existing metaheuristics in order to reduce the expected energy consumption in tasks scheduling problems with stochastic processing times.

In particular, among the different scheduling problems existing in the literature, we will focus on the well-known Flow-Shop Problem (FSP) [50]. Notice, however, that simulation-based approaches similar to the one considered in this work could be employed in solving other scheduling variants with stochastic components.

3.3.2 Flow-Shop Problem with Stochastic Times related work

The literature on the Flow Shop Problem with Stochastic Times (FSPST) is not as extensive as the deterministic case. Pinedo modeled the permutation flow-shop problem with and without intermediate storage [126], where the processing times of a given job on the various machines were independent realizations from the same probability distribution and the objective was to minimize the expected makespan. Dodin considered different probability distributions for processing times [127]. He assumed these processing times were independent and not identically distributed random variables and his goal was also to minimize the expected makespan.

Gourgand et al. used recursive algorithms, based on Markov chains, to compute the expected makespan in conjunction with a simulation model as a backup to evaluate the expected makespan [128, 129]. Wang et al. proposed Genetic Algorithm (GA) approaches for minimizing the expected makespan when processing times follow a Uniform distribution [130, 131].

More recently, Baker and Trietsch developed heuristics for the two-machine flow-shop problem where the processing times are independent random variables following a known probability distribution [132]——also with the objective of minimizing the expected makespan. Some authors also studied different variations of the flow-shop problem: Allaoui et al. [133], Choi and Wang [134] and Kianfar et al. [135] considered the stochastic hybrid flow-shop scheduling problem. Zhou and Cui presented an approach for the multi-objective stochastic flow shop problem [136], on which processing times were normally distributed with two objectives to minimize: flow time and delay time of the jobs.

Most of the aforementioned publications made restrictive assumptions on the probability distributions modeling the processing times of each job-machine pair and the independence and identical distribution of the tasks. Recently, Baker and Altheimer combined simulation and heuristics to solve the FSPST with the objective of minimizing the expected makespan [137]. They created three sets of stochastic flow shop problems where the processing times were distributed following different distributions. The authors proposed three heuristic methods, two of them based on the CDS heuristic [138] and the third based on the well known NEH heuristic [139].

3.3.3 A simheuristic task scheduler

In the stochastic scheduling scenario described, the goal of our proposal is to find a tasks schedule that allows us to reduce the expected makespan and, consequently, the associated energy consumption.

Our approach is inspired in two facts: (a) the FSP is just a FSPST with constant demands —random demands with zero variance; and (b) while the FSPST is yet an emerging research area, extremely efficient metaheuristics do already exist for solving the FSP . Thus, given an initial FSPST instance, our approach proposes to transform the stochastic instance into a deterministic instance by replacing each random variable time by its mean or expected time; and then to obtain a set of high-quality solutions for the deterministic problem by using any of the existing efficient FSP algorithms, e.g. the NEH heuristic [139] or the IG metaheuristic [140].

A solution to the FSP instance is simply a given permutation of tasks, and therefore it is also a feasible solution of the original FSPST instance. Then, simulation is used to determine which solution, among the best-found deterministic ones, produces the lowest expected makespan when considering stochastic times.

This simulation process works as follows. Given a solution (permutation of tasks) to the FSP, the probability distribution of each job-machine processing time is used to generate a random variate; and then, these random variates are used to compute the stochastic makespan associated with the given permutation of tasks. Iterating this process, it is possible to generate a random sample of makespan observations associated with the given solution. From this sample, different statistics for the stochastic makespan can be obtained, including point and interval estimates for the expected stochastic makespan.

3.3.4 Numerical experiments

In order to test how our approach could contribute to reduce expected makespan and energy consumption in a computing facility, we prepared a set of stochastic instances which can be seen as a generalization of the deterministic FSP instances proposed by Taillard[1] [141].

Taillard's instances are grouped in sets of 10, according to the number of jobs (n) and machines (m) considered. Usually, these numbers are represented at the end of the name of each instance in the form $n \times m$. We selected randomly 10 of the whole instance set for our experiments, as shown in Table 3.1.

For each of these deterministic instances, we changed the constant processing times by random variables as follows: if p_{ij} represents the processing time (in hours) of the i-th task in the j-th server, then we assumed this processing time to be a random variable, P_{ij}, following a Log-Normal probability distribution with $E[P_{ij}] = p_{ij}$ and $var[P_{ij}] = k \times p_{ij}$, where $k \geq 0$. In our experiments, we set $k = 5$ in order to consider a scenario with a relatively high variance. At this point, it is important to notice that, even we used a particular probability distribution for generating our numerical experiments, the methodology explained here could be employed with any other probability distribution and parameters.

First of all, our strategy assumes that there exists a strong correlation between solutions for the FSP and solutions for the FSPST. We expect this assumption to be particularly valid when stochastic times show a relatively low variability. To determine so, we ran some preliminary experiments on the studied instances and we obtained a correlation coefficient of 0.99 for all variances with factors $k \in \{0.1, 0.5, 2, 5\}$. These statistics indicate there is a strong correlation between deterministic and stochastic values for the expected makespan. We show in Figure 3.2 the scatterplot with the relationship of deterministic and stochastic makespans of four randomly chosen instances among the ones with 20 servers, since we can show more values on these instances and so depict larger plots.

As can be observed, even when the time deviation is substantial, there is a clear correlation between expected makespans for the FSPST and deterministic makespans for the FSP. However, the best-found FSP solution —the one with the lowest deterministic makespan—

[1]Available online at `http://mistic.heig-vd.ch/taillard`

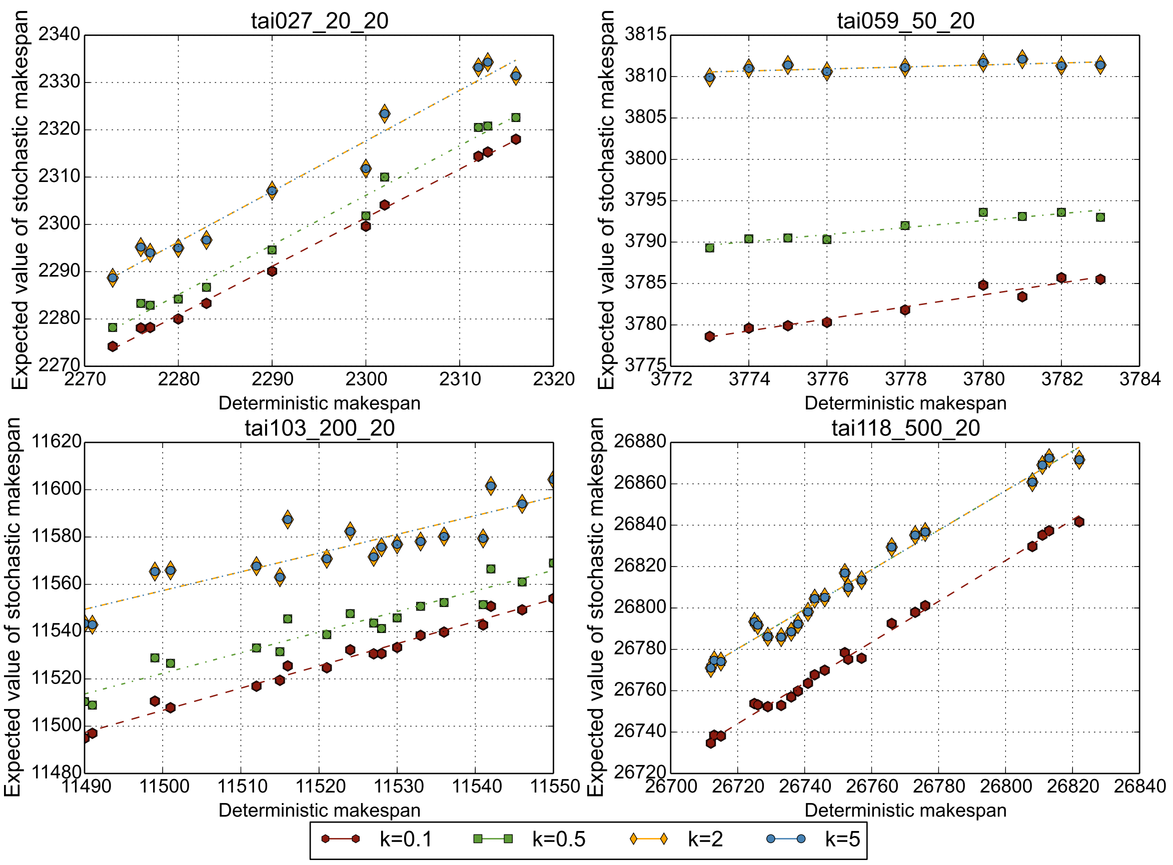

Figure 3.2: Correlation assumption tests in instances `tai27`, `tai59`, `tai103`, `tai118`. All axes in time units.

will not necessarily be the best-found FSPST solution —the one with the lowest expected stochastic makespan.

We then implemented our algorithm as a Java application. Even though Java may run slower than other compiled languages (such as C or C++), it offers some advantages like the fact that it is platform independent as well as that it facilitates repeatability of results [142]. We performed all the tests in a computer based on an Intel Xeon at 2.0 GHz with 4 GB RAM running Windows 7 and the Java Virtual Machine 7-64-bits.

Table 3.1 shows the results obtained by applying our simulation approach in combination with several scheduling algorithms: (a) Sim-Random refers to using a random order of tasks; (b) Sim-NEH refers to using the order provided by the NEH heuristic; and (c) Sim-IG refers to using the order provided by the IG metaheuristic.

Figure 3.3 shows a comparison among the different scheduling strategies and the savings

Table 3.1: Makespan and energy consumptions among different scheduling policies

			Expected makespan [hours]			**Expected energy consumption [kWh]**			**Expected savings [%]**	
Instance	**# tasks**	**# servers**	**Sim-Rand**	**Sim-NEH**	**Sim-IG**	**Sim-Rand**	**Sim-NEH**	**Sim-IG**	**NEH-Rand**	**SimIG-Rand**
tai007	20	5	1,535.8	1,287.6	1,252.7	6,927.8	6,437.9	6,263.5	-16.2%	-18.4%
tai011	20	10	2,012.8	1,696.7	1,613.3	18,156.4	16,966.8	16,133.2	-15.7%	-19.8%
tai036	50	5	3,2239	2,863.3	2,838.2	16,248.4	14,316.6	14,191.0	-11.2%	-12.0%
tai045	50	10	3,749.7	3,157.7	3,041.8	34,218.8	31,577.2	30,418.1	-15.8%	-18.9%
tai067	100	5	5,988.7	5,340.9	5,310.4	30,143.4	26,704.6	26,551.8	-10.8%	-11.3%
tai087	200	20	8,048.1	6,695.1	6,398.3	146,701.6	133,902.4	127,966.4	-16.8%	-20.5%
tai097	200	10	12,844.0	11,034.2	10,934.1	121,655.2	110,342.2	109,341.0	-14.1%	-14.9%
tai104	200	20	13,494.0	11,738.7	11,448.7	258,384.4	234,774.0	228,973.6	-13.0%	-15.2%
tai112	500	20	31,242.2	27,399.9	26,933.0	602,051.0	547,997.8	538,660.2	-12.3%	-13.8%
tai118	500	20	31,070.6	27,383.0	26,890.2	593,882.8	547,660.0	537,803.0	-11.9%	-13.5%
Averages									-13.8%	-15.8%

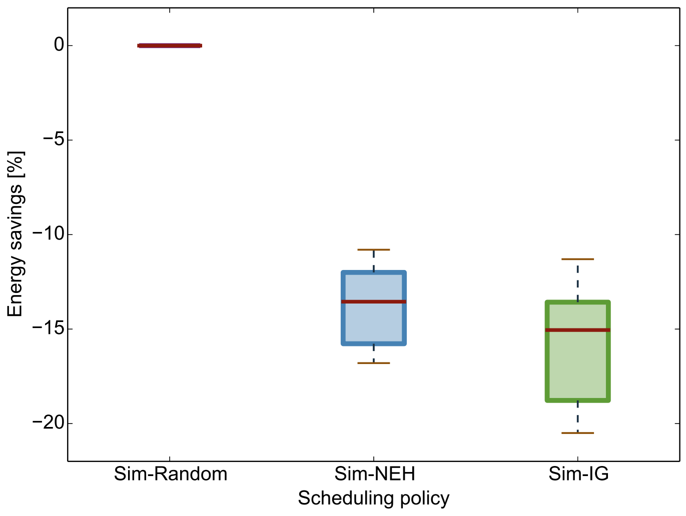

Figure 3.3: Boxplot comparison of the energy savings with each scheduling policy

each of them can provide. The following conclusions can be extracted for the sample set of instances considered:

- By using an extremely fast heuristic like the NEH, it is possible to obtain in real time —less than a second for the size of the instances considered— a tasks schedule which is able to generate noticeable savings (about 13.8% on the average) in both expected makespan and expected energy consumption.

- These savings can be increased even further (up to a 15.8%) by employing a state-of-the-art metaheuristic, which employs just a few seconds in generating a near-optimal tasks schedule.

The results of these experiments show remarkable reductions, both in expected makespan as well as in energy consumption, with regards to using a random scheduling strategy.

3.4 Green computing challenges associated with non-dedicated resources

We discussed in Chapter 2 the importance of service availability as an incentive to attract new users and services into contributory platforms. We presented a methodology framework to allocate composite services in a contributory system that guarantees a certain availability level while minimized the deployment cost.

Besides the service availability, it would be profitable to reduce the energy consumption of every service deployed in a community. Energy consumption of every single host when supporting a given service could be combined with the availability when selecting the hosts to place service replicas and so obtain highly available and less energy-consuming service deployments.

In this section, we devise a heuristic-based methodology to efficiently and automatically select the set of hosts that provides a given level of service availability while minimizing the energy consumption of the deployment. This proposal is based on employing the system availability estimator used in Chapter 2 in combination with an elementary but much faster heuristic for host selection.

3.4.1 Problem description

We adapted the methodology in in Chapter 2, so the goal is to long-lived services on top of non-dedicated resources for a period of time $\mathcal{T}$, after which some component re-assignation might be required. This is repeated until the service is explicitly stopped by an external agent. We assume all services are either stateless or has an internal procedure to maintain status amongst different replicas and a short processing time for queries, at least compared with time $\mathcal{T}$. Thanks to the former, service instances are easily replicable and many of them might be deployed simultaneously, offering some redundancy to the system.

We assume that a distributed service is a set $\mathcal{P}$ of interrelated processes that must run for a (usually long) period of time $(0, \mathcal{T})$, as described in Section 1.3.2. Let us assume that we have a pool $\mathcal{N}$ of n available hosts to deploy the service. Each of them is characterized by a certain availability behavior and an energy consumption model.

Host availability matters were discussed earlier in Chapter 2. About the energy consumption of each host, consider it to be a variable with many factors that affect this value and its great variability over time. Despite this, some authors proposed to model the energy consumption of a computer as a linear function directly proportional to the load of the computer [116]. Therefore, it can be approximated as $e_{n_i}(t) = e_{n_i}^{min} + (e_{n_i}^{max} - e_{n_i}^{min}) \cdot s_{n_i}(t)$, where $e_{n_i}^{min}$ is the energy employed when there is no load in that computer, $e_{n_i}^{max}$ is the energy consumed at maximum load in the computer and $s_{n_i}(t)$ is the percentage load of the host n_i at time t. Consider $e_{n_i} \in [e_{n_i}^{min}, e_{n_i}^{max}]$ for $s_{n_i} \in [0, 1]$. This energy model is valid if we assume resources incorporate effective power-saving mechanisms, such as DVFS in modern processors

Apart from their availability and energy consumption, we consider all nodes can host any process, but only one process at a given time. Clearly, a process is available at a given time t if and only if its host node is available at time t. A *deployment* of any distributed service is an injective function $\mathcal{D} : \mathcal{P} \rightarrow \mathcal{N}$. For the sake of simplicity we use the expression D_j to indicate the service deployed over a set of given hosts. Another simplified description of the deployment is given by a set of binary variables $x_{n_i} \forall i \in \mathcal{N}$, where $x_{n_i} = 1$ if the node n_i was selected to host some process of the service, and $x_{n_i} = 0$ otherwise. The energy associated with the deployment is the sum of the individual energy consumption of the selected hosts and this is the parameter that we want to minimize.

As services are something continuous on time, we attempt to provide a certain level of availability by guaranteeing the service will be available a percentage of the time $\mathcal{T}$. This percentage (a_{target}) must be one of the service requirements given when specifying the service itself. In summary, our problem can be formulate as a discrete optimization problem with a restriction:

$$\begin{array}{ll} \text{Find} & D : \mathcal{P} \rightarrow \mathcal{N} \\ \text{that minimizes} & \sum_{\forall i \in \mathcal{N}} x_{n_i} e_{n_i} \\ \text{subject to} & \bar{a}_D \geq a_{target} \end{array}$$

As in the problem description in Section 2.2, the scale of the system has a great impact on the complexity of the problem. In this case, the problem can also be modeled as a *k-item Knapsack Problem* (*k*KP) [58, 59]. No particular method is required to obtain $\bar{a}_D$, so it would be possible to use analytical exact methods or simulation tools (as earlier explained in Section 2.4.1).

3.4.2 Methodology

Because of the size of the search space for service deployments in contributory communities, we developed a fast heuristic to determine pseudo-optimal deployments in restrained times. We assume historical information of the host availability and unavailability intervals; a mechanism to continuously monitor the load of the involved hosts at any given time; the target services are coherent (that is $\bar{a}_{D_j} > \bar{a}_{D_{j+1}}$); and a user service deployment request that indicates a desired availability level a_{target}. We obtain $\bar{a}_D$ using the simulation method explained earlier in Section 2.4.1. Our new contribution is a host selection methodology that follows the next steps:

1. Order the list of available hosts by mean availability in descending order, in such a way that $\bar{a}_{n_i} > \bar{a}_{n_{i+1}}$.

2. From the host ordered list, obtain a list of possible service deployments, considering only host subsets of consecutive elements in the host list. If service size is m and n hosts are available in the system, the obtained list should contain $n-m+1$ deployments. This step is possible due to the coherent traits of the studied systems (that is $\bar{a}_{D_j} > \bar{a}_{D_{j+1}}$).

3. Perform a binary search over the deployments list, keeping the immediate deployment in list such that $\bar{a}_{D_j} \geq a_{target}$. We named this deployment as D_{limit}.

4. Perform a linear search upwards the availability-sorted list of deployments. Since we consider the services to be coherent, all the deployments in this part of the list would have $a_{D_j} \geq \bar{a}_{target}$, but the energy consumption associated to the selected hosts may be lower ($e_{D_{limit}} \geq e_{D_{limit-j}}$). Therefore, we seek for the deployment offering the lowest energy consumption only in that part of the list, since all deployments on it will fulfill the availability requirement.

Figure 3.4 shows a graphical chart of the operation of our proposal. Data used in the figure does not belong to a real scenario and numbers were only chosen to clarify the example. On it, D_{limit} is D_2, but the energy consumption of D_1 is lower, so the chosen deployment should be the latter (which still fulfills the availability requirements).

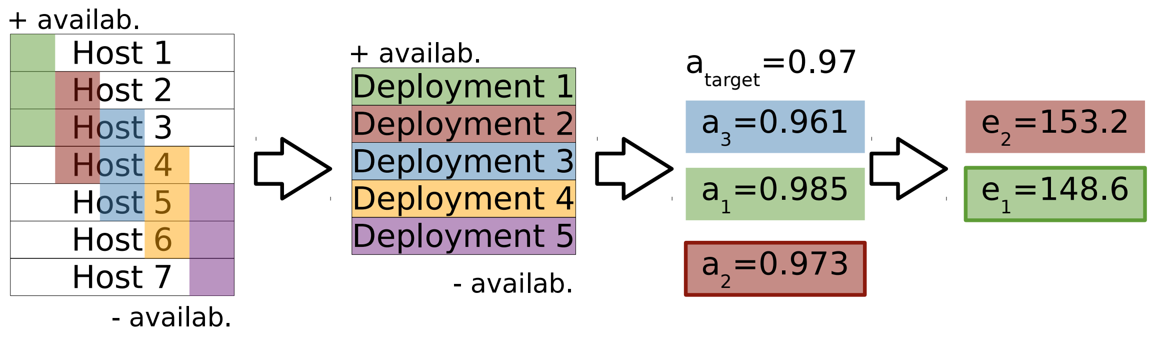

Figure 3.4: Sketch of the proposed methodology in a reduced scenario: three-component service and seven available hosts in the community

This methodology is flexible enough to adapt to very different service topologies and host behaviors. As well, it is ready to work either on large or small scale communities. Due to the stochastic nature of the load on the different hosts involved in a community, the pseudo-optimal deployments obtained by our heuristic might vary for a given service, since it depends on the system status at time the request was done.

3.4.3 Numerical experiment

As far as we know, there exist no real traces of availability and energy consumption the systems such as the ones we are dealing with. For this reason, we syntheticallyy generated the information of 10, 000 hosts with the parameters shown in Table 3.2. `RandUniform(a,b)` stands for a random number generation function following a Uniform distribution within the [a,b] range.

Node ID		Node i
Failure Distribution	**Type**	Weibull
	Shape	`RandUniform(0.8, 2)`
	Scale	`RandUniform(0.5, 4)`
Repair Distribution	**Type**	Weibull
	Shape	`RandUniform(0.5, 1.75)`
	Scale	`RandUniform(0.2, 1.8)`
$\bar{\mathbf{a}}_{\mathbf{n_i}}$		$\frac{E[failure_i()]}{E[failure_i()]+E[repair_i()]}$
$\bar{\mathbf{e}}_{\mathbf{i}}$	$\mathbf{e_{min_i}}$	`RandUniform(20, 50)`
	$\mathbf{e_{max_i}}$	`RandUniform(350, 1000)`

Table 3.2: Synthetically generated historical host information

We then considered an abstract service defined by the topology shown in the directed graph in Figure 3.5. The numbers indicate the arbitrary order in which the hosts are selected from the list ($\bar{a}_{n_i} \geq \bar{a}_{n_{i+1}}$). We chose this particular topology because its simplicity helps to better understand the shown example.

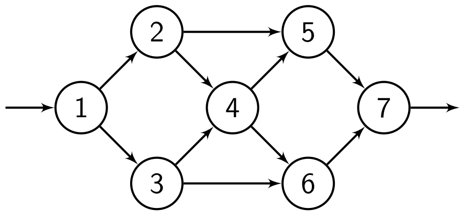

Figure 3.5: Service topology description

We then built a computer simulator in Java to generate random behavior for the load in each of the involved hots if the service was placed on it. Our simulator generated random numbers in the range [0,1] as the load of each computer if the service was deployed on it every time a service deployment request was placed. From the expected load and the energy consumption information describing each host (previously obtained from Table 3.2), it was possible to determine the energy consumption for all the hosts. The energy consumption of a given service deployment can be then obtained by aggregating the energy consumption of all the selected hosts.

We ran the simulator for 100 lifetimes, as if 100 services were to be deployed in the community. In all cases, we fixed the availability requirement at 90% and we recorded the availability and the energy consumption of the most greedy deployment (selecting the most available hosts, as was D_1 in Figure 3.4) and the ones of the deployment found by our methodology. We show the mean results and the mean differences in Table 3.3 and we declare the mean execution time was 903.5 milliseconds in a desktop computer built of an Intel Core i5-2400 processor, 4 GBytes of RAM memory, running Ubuntu Linux 12.10 and the Oracle Java Virtual Machine 7u17-64bits.

	$\bar{\mathbf{a}}_\mathbf{D}$	$\bar{\mathbf{e}}_\mathbf{D}$
Greedy deployments	0.992	4437.33
Our deployments	0.959	1938.35
% gap	-3.31%	-56.32%

Table 3.3: Overview of the measured results

We show in Figure 3.6 the result of four deployment search processes, selected randomly among the 100 performed in the experiment. The graphs depict the temporal process and the evolution of the service availability and energy consumption of the chosen hosts. The vertical dashed line indicates the time the binary search ends and the high climbing search starts, while the horizontal one signals the availability threshold (a_{target}) imposed by the requester.

We chose to compare our method to an availability-greedy method, since it would be the only one that could guarantee the availability of the service without performing any deep study over the dealt system.

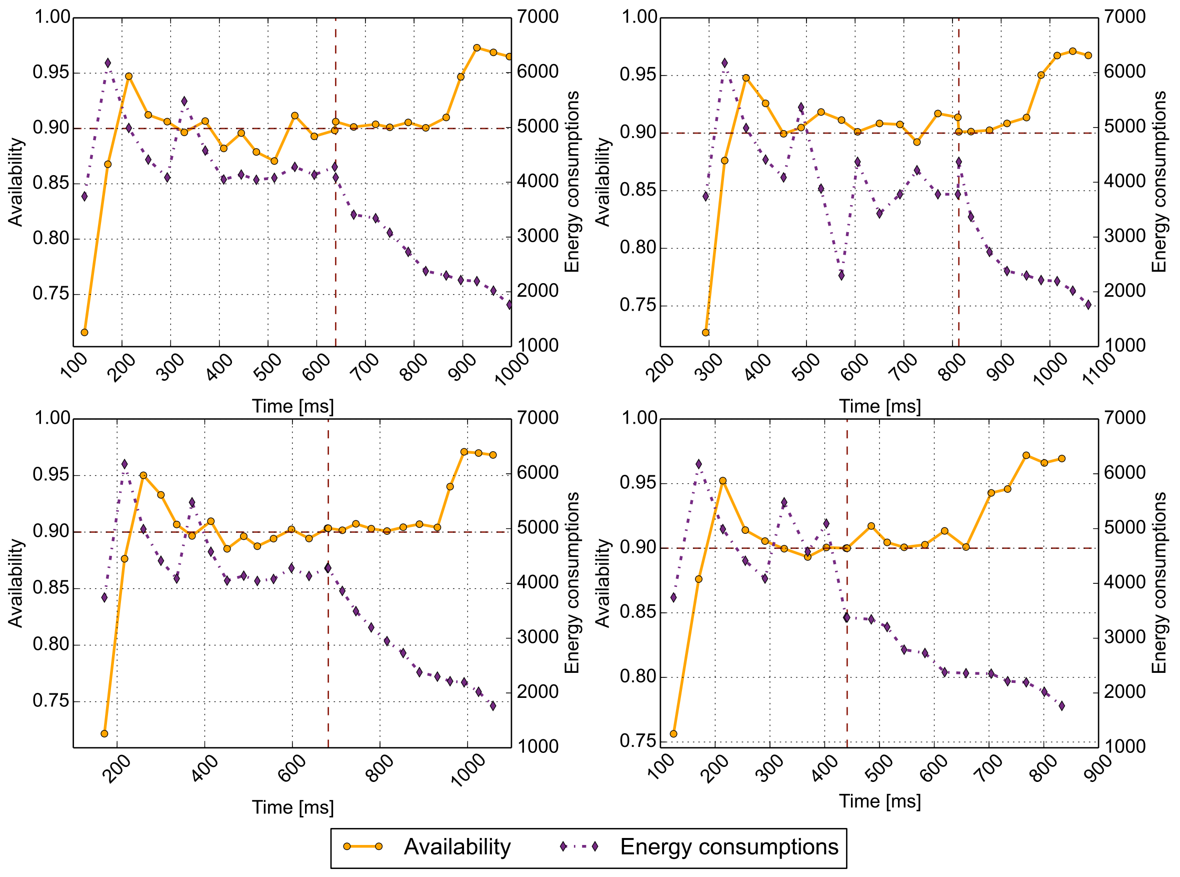

Figure 3.6: Graphical evolution of four different service deployments picked randomly from the 100 ones tested

From the obtained results, we conclude our methodology outperforms the greedy approach in terms of energy consumption while maintains the availability above a given threshold. What is more, our method prove to work fast when dealing with a sizable pool of very different hosts. Thanks to this fact, it could be included in a user-interactive service deployment procedure within a real community.

We evaluated our heuristic by means of computer simulation and we found that investing very few time (less than a second) to perform a search among the available resources, it is possible to reduce the energy consumption up to a 56.32% while degrading the offered availability only a 3.31%.

3.5 Summary

Energy consumption of large-scale Internet distributed computing infrastructures is a matter to be regarded. As the demand for new services and their scale grows at an outstanding rate, its environmental footprints also does. This chapter surveyed the green computing arena related to distributed systems in general.

After that, we have argued that by using optimization algorithms to schedule tasks in servers, energy consumption in these facilities could be reduced in a noticeable way. In our approach, we have considered stochastic processing times as a more realistic assumption than just considering constant ones. Thus, simulation has been combined with existing metaheuristic algorithms in order to obtain numerical values in a set of benchmarks.

We combined as well the environmental concern with the emerging computing concept of contributory communities. We proposed a simple and fast heuristic to determine a subset of hosts suitable to provide a given level of service availability. In addition, we included the energy consumption as a variable in the equation. Thus, we aim at minimizing the energy consumption of the deployed services while maintaining the service availability over a specified threshold.

We showed with both methodologies in this chapter that it is possible to reduce the energy consumptions of distributed computing systems devoting very little efforts to select properly the resources used by services or applications run on them.

4 Network-aware allocation for services in community systems

This chapter proposes a methodology to allocate services in a contributory platform in such a way the network distances to their clients are minimized. Thus, we formalize the problem as a Facility Location Problem. We present a method to effectively solve this classical optimization problem and prove its utility in a real-life example.
The contents of this chapter were extracted from publications [P6, P12, P13, P14].

4.1 Overview

All the allocation methods earlier proposed in this dissertation took into account some traits of the hosts selected to create the service overlay. In all these cases, it was assumed there was a network connection between all the components, but no information regarding the underlying network was used to decide where to allocate services.

In typical Internet applications based on the client-server paradigm, network distance between both actors has a great impact on the quality perception, the overall bandwidth consumption and the network latency. For example, video streaming services might be affected if congestion is found in the path from the server to the final client. Distributing services across the network is a good strategy to reduce these phenomena, but it comes at a high cost for operators. Data-intensive applications can also suffer from degradation and can generate high bandwidth demands if data is stored far from the processing spot. These scenarios fostered the description of the edge computing concept [143], on which analytics occur at the source of the data, which is usually at the edges of the network.

Service providers often deliver their Internet services through well-established Content Delivery Networks (CDN) [144], either their own ones or hiring infrastructure to a third party. These infrastructures distribute their datacenters worldwide, in such a way users are served from their closest digital warehouse. Thus, network delays can be reduced, less overall bandwidth is required in the network and the final user obtains a better service at the end.

In contrast to the CDN model with large infrastructures close to the network core, contributory computing systems are composed by nature of geographically distributed nodes, often placed close to the network edge. Hence, selecting the spots to allocate a service or to storage data becomes an important decision to guarantee system performance and proper resource usage.

As stated above, the methods proposed in Chapters 2 and 3 or the one in [32] only considered some traits of the involved hosts. From the network perspective, these methods could be considered as a random selection strategy. Ryden et al. proposed a user-contributed host selection strategy for data-intensive applications in Nebulas [38]. Their proposal was based on bandwidth probes to reduce overall bandwidth usage and data transfers, but it was specifically meant for data-intensive applications and only considered available bandwidths on the paths between hosts, not the actual network topology.

Selecting locations in a network to place a content or a service as close as possible to the consumers can be modeled as a Facility Location Problem (FLP) [51, 52] or a p-median problem [145]. Both the FLP and the p-median are classical NP-hard optimization problems largely studied in the fields of transportation and logistics.

The p-median problem is devoted to locate a set of facilities relative to a larger set of customers in such a way the sum of the shortest distances between customers and facilities is minimized. It does not consider any opening costs for facilities and restricts a priori the number of them to open. The FLP is similar to the p-median problem, but it introduces a cost incurred when opening a facility and does not limit the number of facilities to open. Then, the goal is to determine a set of facilities that minimizes the sum of shortest distances to all customers and the setup costs incurred by the selected facilities. In both problems, the more facilities selected, the closer should be any customer to any of them, but also more resources would be utilized.

Back to the computing arena, distributed systems can be modeled as a FLP or a p-median problem in which facilities are the servers and the customers are the service consumers. Network topology would determine the distances, either considering network hops or assigning weights to the links according to delays or available bandwidths for instance.

Let us imagine a contributory platform meant to support open-access stateless services, as the ones described in Section 1.3.2 and targeted in [32]. Consider the nodes available to host these services as potential facility spots and the ones finally hosting a replica as the open facilities. It would be possible to use the method in Section 2.6.3 to determine the number of needed replicas to guarantee a certain level of service availability [71] and then select this replicas by solving a p-median problem on the network topology. Nevertheless, since the p-median does not consider any setup cost for selecting a node as facility, supporting concurrent services on the same topology would lead to a situation on which the best services deployed are located in the best network spots.

To address the latter issue, a setup cost for all the available hosts related their network position could be established for each available host and then deal with an FLP instead of a p-median problem. In this case, the number of replicas does not have to be known in advance and selecting the most suitable network positions for services implies incurring in higher deployment costs. Balancing deployment costs would make the most suitable network spots to be shared out among concurrent services in the same topology. The disadvantage of this approach is that service availability might not be guaranteed if the FLP results in the selection of fewer hosts than required by method in [71].

We describe in detail in this chapter a novel heuristic to solve the Uncapacitated Facility Location Problem based on biased randomization techniques and an Iterated Local Search framework. We asses the goodness of our methodology in some of the classical benchmarks for the FLP. Finally, we test our methodology to allocate services in a contributory platform on top of a Community Network.

4.2 The Uncapacitated Facility Location Problem

The (uncapacitated) FLP [51, 52] involves locating an undetermined number of facilities to minimize the sum of setup and serving costs. The problem is defined over an undirected strongly connected graph $G = (V, E)$ where V is composed of a subset of customers $C \subseteq V$ and a subset of facilities $F \subseteq V$, and E is a set of edges connecting the nodes in V. Each edge $e \in E$ has an associated cost of using it $c_e \geq 0$, and for all $i \in F$ we are given a facility opening cost $f_i \geq 0$. Furthermore, for every facility i and customer j we have an associated cost of connecting the customer to the facility $c_{ij} \geq 0, i \in F, j \in C$. Under these circumstances, the objective of the problem is to open a subset of the facilities in F and connect each customer with an open facility, so that the total cost is minimized, as in Equation 4.1.

$$\text{Minimize} \quad \sum_{\forall j \in C} c_{ij} + \sum_{\forall i \in F} f_i \tag{4.1}$$

The uncapacitated FLP is considered as the 'simple' facility location problem [146], where both the alternative facility locations and the customer positions are considered discrete points in the graph. An example of a FLP problem instance can be found in Figure 4.1. This assumes that the alternative sites have been predetermined and the demand in each customer zone is concentrated at the point representing that region. FLP focuses on the production and distribution of a single commodity over a single time period, during which the demand is assumed to be known with certainty. The trait of this discrete location problem is the ability to determine the size of each facility without any restriction.

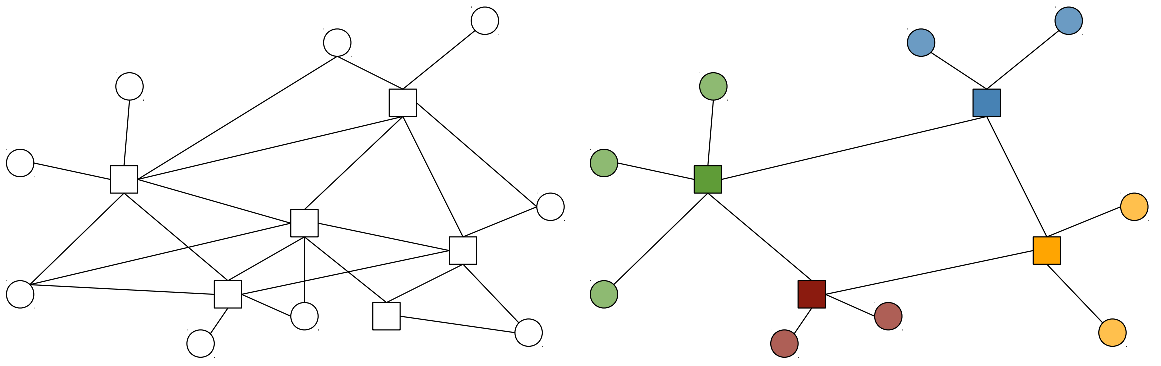
(a) Problem instance (b) Problem solution

Figure 4.1: Example of FLP instance, on which squares are facilities and circles are customers.

Every customer has an associated demand that needs to be served by the facility, d_j. Notice that this demand would take place in the problem definition for the case of the Capacitated Facility Location Problem (CFLP), where every facility has a limited facility, so it is required a constraint to avoid surpassing that facility capacitated. Even without this capacity constraint, the FLP is proved to be NP-hard [147]. For a given instance and a given non-empty subset of facilities $X \subseteq F$, a best assignment $\sigma : C \rightarrow X$ satisfying (Equation 4.2a) can be computed easily. Therefore, we will often call a nonempty $X \subseteq F$ a feasible solution, with facility cost $c_F(X)$ (Equation 4.2b) and service cost $c_S(X)$ (Equation 4.2c).The task is to find a nonempty subset $X \subseteq F$ such that the sum of facility cost and service cost is minimized.

$$c_{\sigma(j)j} = min_{\forall i \in X}(c_{ij}) \tag{4.2a}$$

$$c_F(X) = \sum_{\forall i \in X} f_i \tag{4.2b}$$

$$c_S(X) = \sum_{\forall j \in C} min_{\forall i \in X}(c_{ij}) \tag{4.2c}$$

The FLP can be also formulated as an Integer Program (IP) [148]. In Equation 4.3, assume x_{ij} represents the amount of flow from a facility i to a customer j, which would be 0 if the customer will not be served by that facility, or equals to the demand of the customer otherwise; y_i is a decision variable which is equal to 1 if the facility i will be opened and 0 otherwise. Next constraints force customers to be assigned only to open facilities and assure every customer's demand is satisfied.

$$
\begin{aligned}
&\text{Find} && D : F \to X \\
&\text{that minimizes} && \sum_{\forall i \in F} f_i y_i + \sum_{\forall i \in F} \sum_{\forall j \in C} c_{ij} x_{ij} \\
&\text{subject to} && x_{ij} \leq y_i, \forall i \in F, \forall j \in C \\
& && \sum_{\forall i \in F} x_{ij} = d_j, \forall j \in C \\
& && x_{ij} \geq 0, \forall i \in F, \forall j \in C \\
& && y_i \in \{0,1\}, \forall i \in F
\end{aligned}
\tag{4.3}
$$

4.3 Related work on FLP

The FLP was originally introduced in the early 60's as the Plant Location Problem [51, 52]. It is one of the most common location problems and has been widely studied in the literature, both in theory and in practice [149, 150, 151]. The following sections review some solutions proposed to the problem, as well as different variants of the FLP proposed to cope with different scenarios and its application to computer systems and networks.

4.3.1 Solutions to the problem

The facility location problem has been studied from the perspectives of worst case analysis, probabilistic analysis, polyhedral combinatorial and empirical heuristics [147]. In the existing literature, we can also find exact algorithms for the problem, but its NP-hard nature makes heuristics a more suitable tool to address larger instances. One of the first works on the FLP was a branch-and-bound algorithm [152] that used a compact formulation of FLP to take advantage of the fact that its linear programming relaxation can be solved by inspection. However, this linear programming relaxation does not provide tight lower bounds to the problem.

Another of the earliest approaches proposed for the problem was a direct search or implicit enumeration method [153]. The author defined two different algorithms based on the same directed search, one considering the facilities initially open and a second one considering the facilities initially closed.

Schrage presented a tight linear programming formulation [154] for the location problem different from the one defined in [152]. He applied to this formulation a specialized linear programming algorithm for variable upper bound constraints. Erlenkotter presented a dual-based procedure that begins with this tight linear programming formulation but differed from previous approaches by considering a dual objective function [155]. Körkel presented an improved version of these algorithm in [156].

One of the first non-exact methods proposed to solve the problem was the greedy algorithm proposed by Hochbaum [157]. The first constant factor approximation for this problem was given by Shmoys et al. [158], later improved by Chudak [159]. Still, both of these algorithms were based on LP-rounding and had longer running times. Jain and Vazirani proposed a primal-dual algorithm with faster running times and adapted for solving several related problems [160]. This same algorithm was later enhanced and obtained better results [161]. More recently, Li proposed an improved approximation algorithm that outperformed the former results [162].

Approximation algorithms are very valuable for a theoretical analysis of the problem. However, these algorithms are outperformed in practice by more straightforward heuristics with no performance guarantees when facing more complex problem instances. Constructive algorithms and local search methods for this problem have been used for decades [163]. The authors presented one of the earliest models for the problem and a heuristic procedure solving it. Their heuristic comprised two main phases, first a constructive phase considered as the main program followed by an improvement phase.

Following this work, more sophisticated metaheuristics have been applied to the FLP. Alves and Almeida proposed a Simulated Annealing algorithm that was one of the firsts metaheuristics applied to the problem [164]. Kratica et al. presented a genetic algorithm outperforming previous works [165]. Ghosh presented a Neighborhood Search heuristic for the problem, using Tabu Search as local search and obtaining competitive solutions in very low computational times compared to exact algorithms [166]. Michel and Van Hentenryck defined a simple Tabu Search algorithm [167], which demonstrated to be robust, efficient and competitive when compared with the previous work with genetic algorithms. The Tabu Search algorithm used a linear neighborhood, which flipped a single facility at each iteration. Resende and Werneck proposed an algorithm based on the Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic [168]. The algorithm combined a greedy construction phase with a local search procedure. It obtained results very close to the best-known solution for a wide range of different instance sets. More recently, Lai et al. presented a hybrid algorithm based on Benders' decomposition algorithm and using a genetic algorithm instead of the costly branch-and-bound method to obtain good suboptimal solutions [169]. The computational results indicated that the algorithm was effective and efficient. However the author only compared the performance with the Benders' original algorithm.

Finally, some work has been presented regarding parallel computing. Wang et al. presented an adaptive version of a parallel Multi population Particle Swarm Optimization (MPSO) implemented with OpenMP [170]. The implementation obtained an important improvement in terms of execution times while obtaining competitive results with a standard computer.

4.3.2 Problem variations

Cooper studied the problem of deciding locations of warehouses and allocation of customers demand given the locations and demands of customers [171], which can be considered as the basic facility location problem. After that, many variations of the basic facility location problem have been studied. The first variation defined is by adding a capacity constraint to each of the facilities in the problem, which results in the Capacitated Facility Location Problem (CFLP).

Another immediate generalization of the original FLP is the problem where the delivery of different products is considered. The Multi Commodity Facility Location Problem (MCFLP) studied the problem without any restriction on the number of products at each facility [172]. In the Facility Location with General Cost Function (FLP-GCF), the facility cost is a function on the number of clients assigned to the facility. An additional variant for the problem is the Online Facility Location (OFLP), in which the demand points arrive one at a time and the goal is to maintain a set of facilities to serve these customers [173]. Carrizosa et al. presented a nonlinear variation of the problem where they modified the classical Integer Programming formulation of the problem by adding to the cost a nonlinear function depending on the number of open facilities [174], named as the Nonlinear Minsum Facility Location Problem (NMFLP).

An interesting field of study of variations to this problem are those proposals defined under uncertainty [150], introducing wide variations on any of the parameters of the problem (mainly cost, demands or distances). The goal in these problems is to find a solution that performs well under any possible realization of the random parameters, which means a robust solution. The random parameters can be either continuous or discrete. As an example, Balachandran and Jain presented a CFLP model with piecewise linear production costs that need not be either concave or convex [175]. Demands are random and continuous, described by some joint probability distribution. In this kind of problems, only first-stage decisions are available, so there are no recourse decisions. So, once the locations are set, they cannot be changed after the uncertainty is resolved. The objectives therefore include the expected recourse costs.

Finally, it is worth to mention an extension of the FLP where it is combined with another optimization problem, the Steiner Tree Problem (STP). As a result, Karger and Minkoff defined the Connected Facility Location Problem (ConFLP) [176]. The ConFLP introduces an additional constraint to the problem, which is that a Steiner tree must connect all the open facilities. This variation of the problem combines location and connectivity problems, which is suitable to model different access network design problems.

4.3.3 Applications

Recently the FLP problem found several new applications in digital network design problems. One example is the equipment allocation in Video on Demand (VoD) network deployments [177]. VoD services are complex and resource demanding, so deployments involve careful design of many mechanisms where content attributes and usage should be taken into account. The high bandwidth requirements motivate distributed architectures with replication of content. An important and complicated task part of the network planning phase of these distributed architectures is resource allocation. The growth of peer-to-peer networks and the use of mobile devices for accessing the contents have made the problem even more complex. Another example of FLP application is an approach for survivable network design of citywide wireless broadband based on the FLP model [178]. They address two issues: how to locate the Wi-Fi equipment to maximally cover the given demand; and how to connect Wi-Fi equipment to ensure survivable networking on a real case scenario in the city of Dublin (Ohio).

Maric applied the problem to model the location of long-term health care facilities among given potential sites [179]. The objective is to minimize the maximal number of patients assigned to the established facilities. Examples can be also found in the supply chain management area. Brahimi and Khan showed a real case of a company that outsourced part of its warehousing activity to a third party provider, facing the problem of which spaces to rent in third-party warehouses [180].

The Online Facility Location Problem [173] can model a network design problem in which several servers need to be purchased and each client has to be connected to one of the servers. Once the network has been constructed, additional clients may need to be added to the network. In this case additional costs will appear into the problem such as the connection cost of connecting a new customer to the cluster and, if additional capacity is required to accommodate the increase of demand, an additional server should be purchased (which means opening an additional facility).

4.4 An Iterated Local Search heuristic to solve the UFLP

Our proposal is a hybrid approach that combines an Iterated Local Search (ILS) metaheuristic [181] with biased-randomization techniques [182, 183] to deal with the FLP.

We first define a basic methodology to address the FLP problem in its deterministic variant. These methodology should obtain results not far from optima in restrained times. Then, by randomizing some of these steps of the deterministic heuristic, it is transformed into a randomized procedure. Therefore, it can be run multiple times in order to obtain different solutions for the problem. These executions, being independent one from another, can be executed either in sequential or parallel mode.

Notice that it is possible to perform this randomization process without losing the logic behind the deterministic heuristic by using a biased probability distribution in the procedure. To achieve this, the process can give higher weight to the candidates that are the best choices according to the deterministic heuristic.

With this randomized process, alternative feasible solutions of a similar quality are easily generated. Being an algorithm with a single input parameter, our approach represents an interesting and competitive alternative to other state-of-the-art metaheursitcs, which usually require more cumbersome, time-costly and instance-dependent fine-tuning processes. In practice, these metaheuristic approaches are usually harder to implement and often not reproducible. Furthermore, Kant et al. suggest that most efficient heuristics and metaheuristics are not used in practice because of the difficulties that arise when dealing with real-life scenarios [184]. To the best of our knowledge, our proposal is the first that employs biased-randomization techniques to efficiently solve the FLP.

The proposed method, similarly to GRASP [185], relies on a biased constructive procedure to generate new solutions. This class of methods has been less studied in the literature and can offer competitive results when compared to more sophisticated approaches, which are difficult to implement in practice.

Figure 4.2 details in a flow chart the algorithm, which works as follows. First, it loads the problem instance. Then, it generates an initial random solution, which is the starting point for the ILS procedure. To generate this initial solution, the algorithm chooses a random number p_i between $\frac{|F|}{2}$ and $|F|$, and picks randomly p_i facilities to open. After setting up the open facilities, it computes the total cost of the generated solution, which is selected as starting point.

Selecting always more than the half of facilities is not an arbitrary decision and is justified by the fact that removing facilities from a problem solution is computationally less expensive than opening new facilities. When a facility is removed from a solution, updating the solution costs only implies relocating the customers that were assigned to the closed facilities, while the rest of customers assigned to the other facilities remain unmodified. Nevertheless, opening facilities in the solution forces the evaluation of every single customer assignment to its closest facility, since the new open stations may be closer than previous assignments. So, selecting a large portion of the available facilities might lead to a fast convergence to optima.

After this initial solution is generated, a local search procedure is applied to refine the initial solution. We propose two different local search procedures. The first one is simple and very fast (tiny local search), while the other performs a deeper search with slower execution times (deep local search).

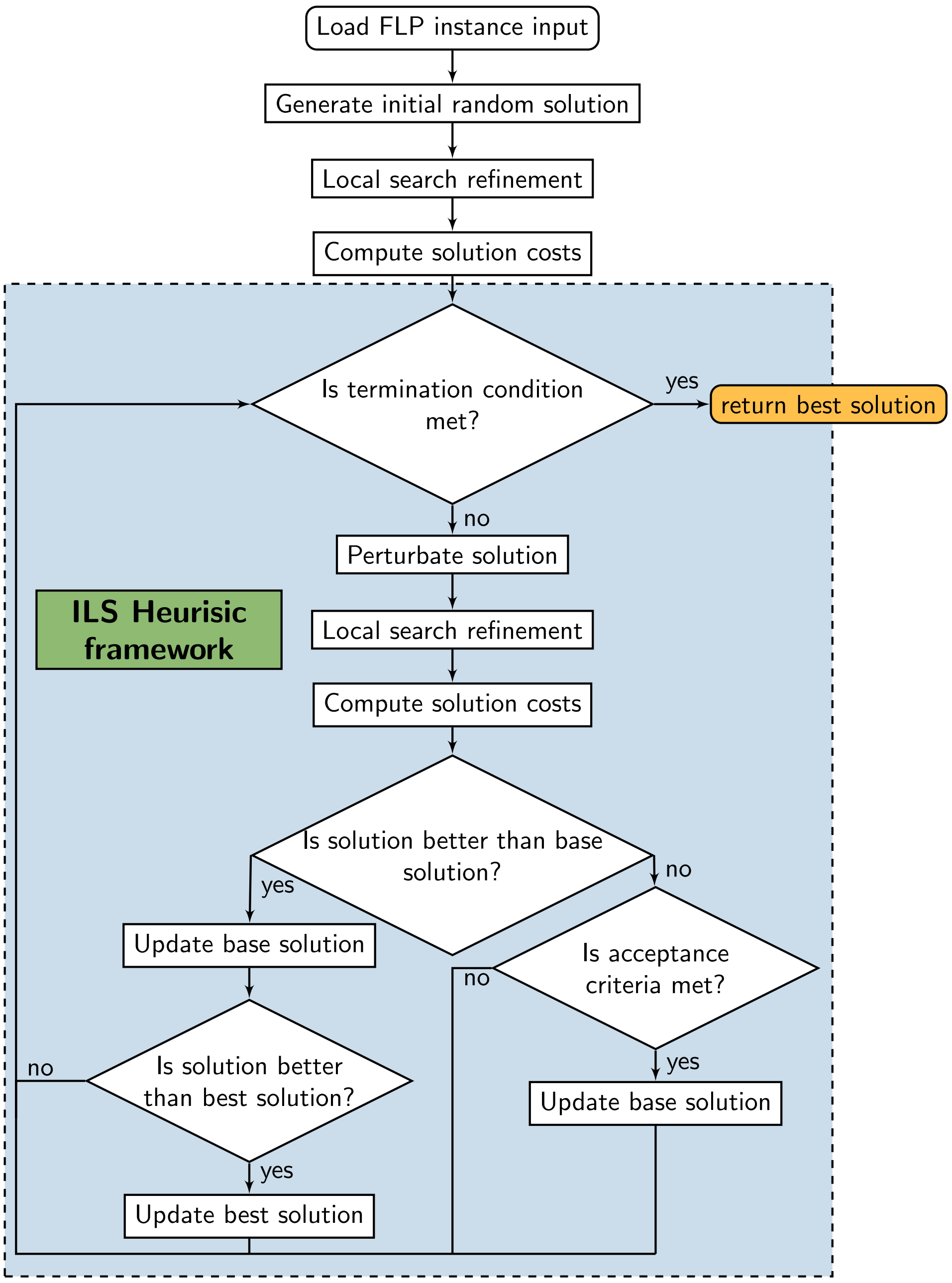

Figure 4.2: Flow diagram of the proposed approach (RandCFH-ILS)

On the one hand, the tiny local search procedure is based only in closing movements. It starts from the current solution and randomly closes one by one each of the open facilities. The facilities to close are selected randomly among all the closed facilities. If the solution is improved (i.e. it has a lower total cost) by closing the selected facility, it is actually removed from the solution; otherwise the facility is kept in.

On the other hand, the deep local search procedure combines both closing and opening movements, which makes this procedure computationally more expensive. This local search procedure is divided in three parts. First, it starts by randomly opening one by one each of the closed facilities. It evaluates the solution at each stage and only adds the open facility to the solution in the case an improvement is obtained. On the second stage it performs a swap movement, replacing a random number of open facilities on the solution by the same number of closed facilities. Finally, in the third stage, it closes the open facilities in the current solution one by one selected in a random order, effectively closing the facility in the final solution only in the case an improvement is obtained. Further details on both local search procedures are detailed in Section 4.4.2.

The solution generated will finally be the starting point for the ILS framework and considered as base solution. The ILS is mainly an iterative process which, at each iteration, generates a new feasible solution with chances to outperform the base solution. Our ILS consists of three steps:

1. Destruction/Construction of the solution (perturbation)

2. Refine the solution (local search)

3. Acceptance criteria of the solution

On the first step, a perturbation operator is applied to the solution. This operator basically destroys some part of the solution by removing open facilities, and then reconstructs it by opening new facilities. This operator always opens more facilities than the amount of closed ones, so it benefits from the fact that the closing movement is less computationally expensive, as happened with the generation of the initial solution. This solution is then refined by the same local search operator used on the initial solution.

Finally, the last step of the ILS is the acceptance criteria for updating the best and base solutions. Being the best solution the one that will be returned as the result of our methodology and the base solution the solution used as initial solution for the next iteration of the ILS. If the solution obtained from the perturbation and local search procedures improves the best solution found so far, then the best and base solutions are updated.

Additionally, if the solution obtained is worse than the current base solution, an acceptance criterion is defined. This acceptance criterion allows a non-improving solution to be accepted as a new base solution if certain conditions are met. The acceptance criteria define a gap which the solution is allowed to worsen. This gap varies during the execution of the algorithm, allowing greater gaps at the beginning of the execution, and smaller gap as the execution time passes. With that, we enable a method to escape from local minimal and explore different regions of the solutions space.

4.4.1 Randomization issues in our approach

The proposed methodology is based on applying biased-randomization techniques at different stages. Biased random sampling can be defined as a set of techniques that make use of random numbers and asymmetric probability distributions to solve certain stochastic and deterministic problems. When properly combined with heuristics, random sampling has proven to be extremely useful in similar stochastic optimization problems [186, 182, 187].

Randomization helps our method to widely explore the space solutions. The first point where we use randomization is the generation of the initial solution. We randomly select a number of facilities which will define the initial solution. The local search procedures also use randomization when exploring facilities, sorting them randomly. Finally, the perturbation operator decides which facilities to remove in the destruction phase and which facilities to include on the construction phase in a random fashion as well.

All these randomization processes must be done without introducing too many input parameters to the algorithm. Otherwise, a non-trivial, time-consuming and often instance-dependent fine-tuning process would be necessary in order to obtain good quality results. The most suitable biased probability functions for these purposes are the single-parameter geometric distribution or the parameter-free decreasing triangular distribution.

When compared to GRASP [185], our approach is not restricting the list of candidates to be evaluated, so we are leaving out one parameter. Still, the randomization is biased, since all the candidate facilities are not considered with the same probability (as shown in Figure 4.3). One of the probability distribution functions which have demonstrated to obtain competitive results with this purpose is the single-parameter geometric distribution [186, 182, 187]. Therefore, the proposed algorithm will use the geometric distribution, which will add a single input parameter to the algorithm (the beta parameter for the probability distribution).

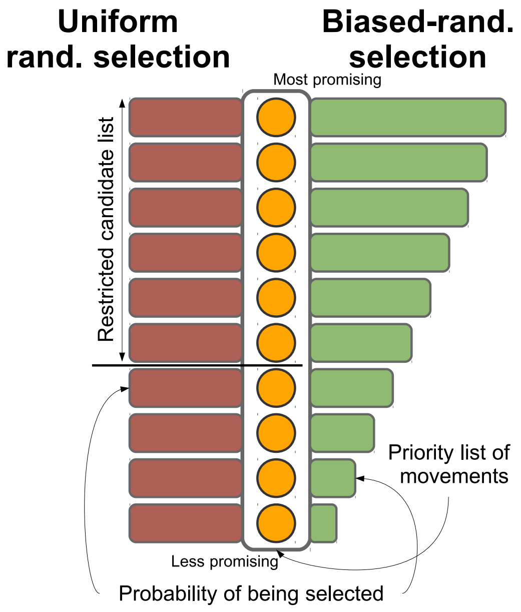
Uniform rand. selection
Biased-rand. selection
Most promising
Restricted candidate list
Priority list of movements
Less promising
Probability of being selected

Figure 4.3: Uniform randomization vs. biased randomization

4.4.2 Pseudo-code of the proposed algorithm

Besides the flow diagram drawn in Figure 4.2, this section describes in detail the operation of all the involved routines in our methodology.

The main method of our proposal is `ILS-UFLP` (Algorithm 3). It receives as parameters the lists of facilities and customers defining the FLP problem instance, the beta parameter required by the geometric distribution used in the randomization process and a parameter used as stopping criterion on the ILS framework.

Our proposal first creates an initial random solution by the `genInitRandSol` method and refines it by the local search procedure. As early mentioned in Section 4.4, we defined two alternative local search procedures: `localSearchTiny` and `localSearchDeep`. After that, credit values used by the acceptance criteria are initialized. Then, the ILS loop is started using the initially obtained solution. In the iterated loop, the `perturbate` operator is applied to the current base solution and the obtained solution is refined again by the chosen local search procedure.

This newly obtained solution is then evaluated. If the solution improves the current base solution, then the base solution is updated. Also the credit is updated with the same value as the improvement obtained. This causes credit values to be higher at the beginning of the execution, so bad solutions are more tolerated. As the algorithm obtains smaller improvements, only small degradations on the base solution are welcomed by the acceptance criteria. If the new solution improves also the best solution, we update it as well.

In the case that the new solution is worse than the current base solution, the difference between both is still evaluated. The new solution is only accepted as the new base solution if this difference is below the credit acceptance threshold. In that case, the new solution is accepted as base solution and the credit is reset to 0, so two degradations in a row are never allowed.

The loop is executed until the termination criterion is met, which can be a time limit or a maximum number of iterations. To conclude, the algorithm returns the best solution found so far.

Once we have detailed the operation of the main method in our proposal, we describe in detail the other methods called from the main procedure in the following paragraphs.

The `genInitRandSol` method (Algorithm 4) is the responsible of generating the initial solution which serves as starting point for the algorithm. It receives as parameters the facilities and customers in the problem instance and the beta parameter required for the Random Number Generator (RNG). It generates a random number between the total number of facilities and the half of this number. Next, it randomly picks this number of facilities from the list of available ones in the problem instance to be open. Finally it constructs the solution with the selected facilities as open facilities and assigns each customer to that facility among the open with lowest service cost for it.

The `perturbate` method (Algorithm 5) describes the perturbation operator used within

Algorithm 3 ILS-UFLP: Iterated Local Search heuristic for the Uncapacitated FLP

```
Require: $facilities, customers, beta, maxIter$
 1: $baseSol \leftarrow genInitRandSol(facilities, customers, beta)$
 2: $baseSol \leftarrow localSearch(baseSol)$
 3: $bestSol \leftarrow baseSol$
 4: $nIter \leftarrow 0$
 5: $credit \leftarrow 0$
 6: while $nIter \leq maxIter$ do
 7:   $newSol \leftarrow perturbate(baseSol, beta)$ {destruction-construction}
 8:   $newSol \leftarrow localSearch(baseSol)$
 9:   $delta \leftarrow cost(newSol) - cost(baseSol)$
10:   if $delta < 0$ then
11:     $credit \leftarrow -delta$
12:     $baseSol \leftarrow newSol$
13:     if $cost(newSol) < cost(bestSol)$ then
14:       $bestSol \leftarrow newSol$
15:     end if
16:   else if $delta > 0$ and $credit \geq delta$ then
17:     {acceptation criterion}
18:     $credit \leftarrow 0$
19:     $baseSol \leftarrow newSol$
20:   end if
21:   $nIter \leftarrow nIter + 1$
22: end while
23: return $bestSol$
```

Algorithm 4 genInitRandSol(): Initial solution generator for the FLP

```
Require: $facilities, customers, beta$
 1: $nFacilToOpen \leftarrow rand(\frac{size(facilities)}{2}, size(facilities))$ {Biased-randomized selection process using a Geometric(beta), promotes the random selection of those facilities with HIGHEST density levels}
 2: $facilToOpen \leftarrow biasedRandSelect(facilities, nFacilToOpen, beta)$
 3: $sol \leftarrow constructSol(facilToOpen, customers)$
 4: return $sol$
```

the main ILS loop. This method receives a base solution and the beta parameter for the RNG. It creates a copy of the base solution and extracts the list of open facilities from it. Then, it randomly closes a random number of facilities to close from all the open ones in the solution. After that, it generates another random number of facilities to be opened, being always greater than the number of facilities previously closed. This forces the new solution to always include more open facilities and benefits the algorithm on the later refinement process of the fact that closing a facility is a cheaper operation than opening a new facility. Once we determine the number of facilities to open, we randomly pick the right amount from the closed list and reconstruct the solution by adding them.

Algorithm 5 perturbate(): FLP solution perturbation

```
Require: $baseSol, beta$
1: $sol \leftarrow copy(baseSol)$
2: $openFacilities \leftarrow getOpenFacilities(sol)$
3: $nFacilToClose \leftarrow rand(0, size(openFacilities))$ {Biased-randomized selection process using a Geometric(beta), promotes the random selection of those facilities with LOWEST density levels}
4: $invOpenFacilities \leftarrow inverseOrder(openFacilities)$
5: $facilToClose \leftarrow biasedRandSelect(invOpenFacilities, nFacilToClose, beta)$
6: $sol \leftarrow destructSol(sol, facilToClose)$
7: $closedFacilities \leftarrow getClosedFacilities(sol)$ {Re-construct more than destructed since closeLS is faster than openLS}
8: $nFacilToOpen \leftarrow rand(nFacilToClose, size(closedFacilities))$ {Biased-randomized selection process using a Geometric(beta), promotes random selection of those facilities with HIGHEST density levels}
9: $facilToOpen \leftarrow biasedRandSelect(closedFacilities, nFacilToOpen, beta)$
10: $sol \leftarrow constructSol(facilToOpen)$ {compute assignment cost}
11: return $sol$
```

The `localSearchTiny` method (Algorithm 6) describes the operation of the tiny local search, which can be used to refine FLP solutions. This procedure receives as parameters the base solution and the beta parameter for the RNG. It creates a copy of the base solution, extracts the list of open facilities and sorts it randomly. After that, all open facilities are removed from the solution one at a time. If the solution without that facility has a lower global cost than the one including it, the facility is effectively removed from the solution. Otherwise, it is kept in the open facilities list.

Algorithm 6 localSearchTiny(): fast local search based on facility closing

```
Require: $baseSol, beta)$
1: $sol \leftarrow copy(baseSol)$
2: $openFacilities \leftarrow getOpenFacilities(sol)$
3: $openFacilitiesSorted \leftarrow biasedRandSort(openFacilities$
4: for all $oFacility$ in $openFacilities$ do
5:     $newSol \leftarrow deleteFacility(sol, oFacility)$
6:     if $cost(newSol) < cost(sol)$ then
7:         $sol \leftarrow newSol$
8:     else
9:         $newSol \leftarrow addFacility(sol, oFacility)$
10:    end if
11: end for
12: return $sol$
```

The `localSearchDeep` method (Algorithm 7) presents an alternative local search procedure, which can be used to refine FLP solutions as the previous `localSearchTiny` method. This procedure receives as parameters the base solution and the parameter for the RNG. It starts by creating a copy of the base solution. Then, it starts a loop structured on three different blocks that will keep running while an improvement is found for the solution.

On the first block, we try to open closed facilities. The list of closed facilities is extracted from the current solution and is randomly sorted. All closed facilities are added one by one to the solution. If the solution with that facility has a lower global cost than the one without it, the facility is effectively added the solution and the improvement control is set to true. Otherwise, it is kept out. The second block of the loop swaps open and closed facilities. It swaps all the open facilities with all the closed ones one at a time until an improvement is found. If any of the solutions with a swap has a lower global cost than the one without the change, the facilities are effectively swapped from the solution, the improvement control is set to true and the swapping process ends. Finally in the third block, we try to remove open facilities calling the `localSearchTiny`, willing to reduce the solution cost. All open facilities are removed from the solution one at a time. If the solution without that facility has a lower global cost than the one including that facility, the facility is effectively removed from the solution and the improvement control is set to true. Otherwise, it is kept in the open facilities list.

4.5 Performance evaluation against well-established FLP benchmarks

To evaluate and assess the performance of the proposed algorithm, several computational experiments were performed. The algorithm was implemented as a Java 7SE application using the LFSR113 random number generator from the Stochastic Simulation in Java library (SSJ) [188]. We performed all the tests on a commodity desktop computer with an Intel Core i5-2400 at 3.20 GHz and 4 GB RAM running Ubuntu GNU/Linux 13.04.

To test the efficiency of the proposed algorithm, four different well-established classes of FLP instances were used [189]. The selected datasets were chosen with the criteria of

Algorithm 7 localSearchDeep(): local search with exhaustive search

```
Require: $baseSol, beta$
 1: $sol \leftarrow copy(baseSol)$
 2: while $improvement$ do
 3:     $improvement \leftarrow false$
 4:     $closedFacilities \leftarrow getClosedFacilities(sol)$
 5:     $closedFacilitiesSorted \leftarrow biasedRandSort(closedFacilities, beta)$
 6:     for all $cFacility$ in $closedFacilitiesSorted$ do
 7:         $newSol \leftarrow addFacility(sol, cFacility)$
 8:         if $cost(newSol) < cost(sol)$ then
 9:             $sol \leftarrow newSol$
10:             $improvement \leftarrow true$
11:         else
12:             $newSol \leftarrow deleteFacility(sol, cFacility)$
13:         end if
14:     end for
15:     $openFacilities \leftarrow getOpenFacilities(sol)$
16:     for all $oFacility$ in $openFacilities$ do
17:         $newSol \leftarrow deleteFacility(sol, oFacility)$
18:         $closedFacilities \leftarrow getClosedFacilities(sol)$
19:         for all $cFacility$ in $closedFacilities$ do
20:             $newSol \leftarrow addFacility(sol, cFacility)$
21:             if $cost(newSol) < cost(sol)$ then
22:                 $sol \leftarrow newSol$
23:                 $improvement \leftarrow true$
24:                 $break$
25:             else
26:                 $newSol \leftarrow removeFacility(newSol, oFacility)$
27:             end if
28:         end for
29:         if $cost(newSol) < cost(sol)$ then
30:             $sol \leftarrow newSol$
31:             $improvement \leftarrow true$
32:             $break$
33:         else
34:             $newSol \leftarrow addFacility(newSol, cFacility)$
35:         end if
36:     end for
37:     $sol \leftarrow localSearchTiny(sol, beta)$
38: end while
39: return $sol$
```

testing the algorithm against instances of small, medium and big size (in terms of facilities and customers included in the graph). We briefly describe next the used sets:

BK a small-sized class introduced by Bilde and Krarup [190]. It includes 220 instances in total divided in 22 subsets, with the number of facilities varying from 30 to 50 and the number of customers from 80 to 100. These instances were artificially generated by the authors, selecting the assignment costs randomly on the range [0, 1000], and opening costs being always greater than 1000.

GAP a medium-sized class introduced by Kochetov and Ivanenko [191]. It consists of three subsets, each with 30 instances: GAPA, GAPB and GAPC, being GAPC to be the hardest instance. These instances are considered to be especially hard for dual-based methods.

FPP a medium-sized class also introduced by Kochetov and Ivanenko [191]. It includes two subsets, each with 40 instances: FPP11 and FPP 17. Although optimal solutions in this class can be found in polynomial times, the instances are hard for algorithms based on flip and swap local search, since each instance has a large number of strong local optima.

MED big-sized class originally proposed for the p-median problem by Ahn et al. [192], and later used in the context of the uncapacitated FLP [193]. Each instance is a set of n points picked uniformly at random in the unit square. A point represents both a user and a facility, and the corresponding Euclidean distance determines connection costs. The set consists of six different subsets with a different number of facilities and customers (500, 1000, 1500, 2000, 2500 and 3000) and three different opening cost schemes for each subset.

In the performed experiments, we tested the algorithm using the two different local search procedures presented. Each dataset was solved twice: one with the algorithm using the `localSearchTiny` procedure and another one using the `localSearchDeep` procedures described earlier in Section 4.4.2. With this, we will be able to compare both procedures and determine which local search has the better performance depending on the features of the solved instance.

The experiments were run by setting a stopping criteria based on a time limit for every problem instance. Every instance will be solved by generating feasible solutions until the termination criteria of time has been met (as sketched in Figure 4.2). These termination criteria will be different depending on the size and the class of instances being solved. So, the greater the instances are, the more time it is given to the algorithm (see Table 4.1). We set these times long enough to guarantee that the algorithm reaches its best solution. We record the time when the best solution was found for every instance, so we can know how fast the convergence to this best solution was.

Table 4.1: Termination criteria for every class of instances

Class	**# facilities**	**Time (s)**
BK	80-100	30
FPP11	133	600
FPP17	307	
GAPA	100	180
GAPB		
GAPC		
MED	500-3000	3600

Next, we show the results obtained when executing our algorithm in the earlier explained benchmarks. We compare our results using both of the local search methods described with the ones declared by Resende and Werneck using their GRASP algorithm [168]. Our experiments were performed on a low-end commodity desktop computer using a standard Java SE application, instead of the specifically compiled application for a supercomputer utilized by Resende and Werneck. Although the theoretically higher performance of our computer, we only used one of its cores for the Java Virtual Machine to run the experiments.

We started evaluating the methodology in the set of tests with simplest and smallest problem instances, the Bilde and Krarup benchmark.

In this test, our algorithm clearly outperforms the GRASP proposal when using the deep local search method. We found the optimal solution for all the instances in the benchmark in much shorter execution times. Contrarily, we discovered in this first test that the tiny local search method obtains lower quality results in terms of gap with the optima in comparable times with the GRASP.

We performed the next evaluation over the GAP benchmark, known as a hard test for

Table 4.2: Results obtained for the 22 subsets of instances in the Bilde and Krarup benchmark

			GRASP		ILS-UFLP tiny		ILS-UFLP deep	
Instance	**# customers**	**# facilities**	**Gap**	**t (ms)**	**Gap**	**t (ms)**	**Gap**	**t (ms)**
B	100	50	0.000	310	0.000	794	0.000	0.20
C	100	50	0.016	450	0.130	835	0.000	0.05
D01	80	30	0.000	223	0.001	116	0.000	0.02
D02				211		317		0.04
D03				199		58		0.06
D04				170		48		0.08
D05				162		41		0.10
D06				186		124		0.12
D07				174		9		0.15
D08				166		20		0.08
D09				175		11		0.07
D10				166		28		0.21
E01	100	50	0.000	476	0.201	667	0.000	0.03
E02				588	0.011	1176		0.07
E03			0.019	512	0.000	286		0.11
E04			0.000	464		389		0.14
E05				376		223		0.18
E06				408		247		0.22
E07				416		451		0.25
E08				418		728		0.28
E09				352		35		0.29
E10				353		70		0.32
Average			0.002	316.14	0.015	290	0.000	0.14

dual-based oriented methods. Neither our algorithm nor GRASP were thought specifically for dual-based problems, so finding good quality results in these instances is challenging for both algorithms.

Table 4.3: Results obtained for subsets of instances in the GAP benchmark

			GRASP		ILS-UFLP tiny		ILS-UFLP deep	
Instance	**# customers**	**# facilities**	**Gap**	**t (s)**	**Gap**	**t (s)**	**Gap**	**t (s)**
GAP A	100	100	5.140	1.41	2.470	47.68	1.095	31.25
GAP B			5.980	1.81	2.811	63.56	1.642	40.72
GAP C			6.740	1.89	2.859	70.17	1.207	59.55
Average			5.953	1.7	3.405	54.43	1.314	43.84

In this set of tests, our algorithm obtained better quality solutions using any of the presented local search methods, although the deep one resulted in much lower gaps from optima. However, the running times employed by our proposal were much higher than the ones in Resende and Werneck's experimentation.

The next test was performed over the FPP benchmark, known to include very hard instances for swapping algorithms. Even though our algorithm is not purely in this family, since we close and open an independent number of facilities at each overcoming iteration instead of directly swapping facilities, this benchmark should also be defiant.

Table 4.4: Results obtained for subsets of instances in the FPP benchmark

			GRASP		**ILS-UFLP tiny**		**ILS-UFLP deep**	
Instance	**# customers**	**# facilities**	**Gap**	**t (s)**	**Gap**	**t (s)**	**Gap**	**t (s)**
FPP 11	133	133	8.480	2.58	0.063	127.126	0.000	111.02
FPP 17	307	307	58.270	25.18	70.731	272.94	12.283	253.44
Average			33.375	13.88	35.397	200.03	6.142	182.22

As in the GAP experimentation, our methodology outperforms the results' quality of the Resende and Werneck's proposal. In the small instance, we even discover the optimal solution when using the deep local search, while GRASP remained at an 8.4% gap. In this benchmark, the tiny local search performs poorly due its simplicity in generating new solutions and the intended complexity of the problem instance. Again, the employed times are much longer than in GRASP.

Finally, we run the MED benchmark, the one with largest problem instances. In these tests, the optimal is not known, but lower and upper bounds obtained with exact methods are provided. We show the gap with respect to the lower bound and compare it to the same gap of the average obtained by the GRASP method.

The results in Table 4.5 show our algorithm performs slightly better (a gap reduction of 0.002%) than GRASP in the instances with larger setup costs (the *-10 instances). In these smallest instances, our running times are even competitive with Resende and Werneck's experiments (almost 200 seconds faster on average). However, on larger instances, our algorithm performs much slower, due to the longer list of open facilities included in these solutions.

Notice some of the times employed to find the best solution are larger than the maximum run time given to the algorithm. This happens in the tests with the lowest facility opening costs (the *-1000 instances). In these problems, the number of open facilities is considerably

Table 4.5: Results obtained for subsets of instances in the MED benchmark

			GRASP		**ILS-UFLP tiny**		**ILS-UFLP deep**	
Instance	**# customers**	**# facilities**	**Gap**	**t (s)**	**Gap**	**t (s)**	**Gap**	**t (s)**
0500-10	500	500	0.022	33.2	0.022	213	0.022	7.35
0500-100			0.016	32.9	0.093	676	0.014	720.2
0500-1000			0.071	23.6	0.071	3168	0.078	1410.5
1000-10	1000	1000	0.101	173.9	0.399	2038	0.099	69.3
1000-100			0.048	148.8	0.333	3566	0.088	1462.9
1000-1000			0.037	141.7	1.165	3468	0.278	3718.2
1500-10	1500	1500	0.191	347.8	0.236	2418	0.175	860
1500-100			0.030	378.7	0.687	3582	0.094	1427
1500-1000			0.034	387.2	3.514	3540	0.320	8074.1
2000-10	2000	2000	0.052	717.5	0.223	3415	0.052	384.6
2000-100			0.036	650.8	1.198	3573	0.299	2201.3
2000-1000			0.031	760.0	5.468	2838	0.400	30088
2500-10	2500	2500	0.164	1419.5	0.622	2988	0.168	496.2
2500-100			0.049	1128.2	1.537	3569	0.349	2347.1
2500-1000			0.052	1309.4	5.964	3533	0.308	50111
3000-10	3000	3000	0.104	1621.1	0.372	3570	0.102	1309.5
3000-100			0.124	1977.6	1.707	3538	0.482	1974.8
3000-1000			0.043	2081.4	6.238	3427	0.319	72252.4
Average			0.067	740	1.659	2951	0.203	9939.7

large and our algorithm tries to improve iteratively a long list. Thus, a single iteration takes longer than the time set for the stopping criteria.

In most of the employed benchmarks our algorithm performed slower than the state-of-the-art heuristic from Resende and Werneck. The reason behind these longer times is the iterative nature of our methodology. On those scenarios on which the list of open facilities is quite long, the algorithm has to iterate over all the elements and perform the earlier explained operations, which takes longer times than the operations in GRASP. Nevertheless, the use of biased-randomization techniques drive the heuristic to better solutions of the problem.

With all the shown benchmarks, we can conclude our methodology outperforms the existing state-of-the-art heuristics in small and medium-scale scenarios, especially on those with short lists of open facilities. In large instances, our proposal is competitive in the case the facilities setup costs are large in comparison to connection costs. Hence, our algorithm could be a valuable tool for reduced or clustered scenarios, on which a large amount of clients could be served by a small amount of facilities, as is the case in access networks in urban areas.

4.6 Open-access services in a Community Network

We validated the suitability of the proposed methodology for network-aware service allocation in a real scenario of a community network. We selected a mesh network topology from Guifi.net, a successful community network. The reason for choosing this network was the shared philosophy of a user-contributed system and the public availability of the network topology (an often hidden information for current commercial access networks).

4.6.1 Guifi.net, a successful Community Network

Guifi.net is one of the largest and most successful Community Networks of the world. It currently connects up to 24,652 operative nodes and 28,689 links[1] in a mesh topology. These topology is divided into more than 130 administrative divisions, each of them containing some distributed monitoring facilities that help to control the correct network operation.

Most of the links in the network are wireless, although fiber optics are used in some key regions of the network. The network is mostly based in the Catalunya region, although other communities in different regions around the world are also growing their local deployments and join the original network through Internet tunnels.

The network was born as a user initiative to connect themselves by creating a free and independent telecommunications infrastructure. Later, the Guifi.net Foundation was created to guard the network freedom and reinforce the infrastructure on its core equipment. This foundation takes care of maintaining the network operative, provides some basic capabilities and organizes actions to improve the network quality or expand it to new regions.

One of the most interesting traits of Guifi.net is that the foundation publicly releases information about nodes, links, network topology and its usage. This poses new research opportunities for researchers, as this information is usually hidden in commercial networks operated by ISPs. A clear sign of this research interest is the existence of research projects devoted to study and improve community networks [194] by designing new protocols and to create a computing platform similar to clouds with the computing resources in these networks [195].

[1] June 2014

4.6.2 FLP in a Community Network

In order to perform our study over a real network, we obtained a complete network snapshot from April 2013 in the Guifi.net Foundation website. These snapshot contained 8,907 operative nodes and 9,047 links divided into 129 administrative regions. For our study, we selected a portion of these network snapshot with 1,134 nodes and 1,161 links. Due to the low visibility of such a large graph, we cannot depict the actually selected network topology. Instead, we provide some basic graph statistics in Table 4.6 to help the reader better understand the type of network we dealt with.

Table 4.6: Traits of the studied network graph

	avg.	**min.**	**max.**
Network diameter			11
Node degree	2.0476	1	260
Node degree centrality	0.0018	0.0009	0.2295
Node closeness centrality	0.2442	0.1212	0.4258
Node betweenness centrality	0.0028	0.00	0.7912
Path lengths	4.2244	1	11

We considered the 53 nodes in the topology with more than one link to be the nodes that could host a service (the facilities in the FLP terminology) and all of them (including the facilities) to be potential service consumers (the clients in the FLP). Considering facilities to be also consumers is the usual case in the community cloud and contributory communities proposals, as users originally contribute their resources to support the platforms with the only reward of accessing these services.

In this example we assume complete knowledge about the network topology. If this was not the case, probing techniques like *traceroute* could be used in a commercial to discover the underlying network connecting the nodes, in a similar way than [38] did with available bandwidth used by their allocation strategy.

To transform the network graph into a classical FLP instance, connection costs were established as the number of network hops from one node to another and the opening cost for each facility was defined as $c_i = 1000 \times closeness\_centrality_i$. The closeness centrality is a graph measure on how close is a vertex to all other vertices in a graph. Thusly, directly linking the opening cost of a facility at a given spot to this measure intuitively seemed a good strategy to associate higher costs to well connected hosts.

We studied the application of our allocation methodology based on the FLP in the described scenario and compared it to the following allocation strategies over the same topology:

1. A greedy method that selects the top N facilities ordered by descending opening cost. This method selects the best nodes in terms of closeness centrality, meaning higher setup costs but lower connection ones.
2. A greedy method, that selects the top N facilities ordered by ascending opening cost. This method selects the worst nodes in terms of closeness centrality, meaning lower setup costs but higher connection ones.
3. A random selection of N nodes from all the available facilities. This could be interpreted as any of the other methods presented in previous chapters that did not consider any network parameter.

In all cases, we set N to the same number of nodes selected by our FLP solving method, so the distance and cost comparison is done with the same level of resource usage. For the random selection strategy, we took the average from 100 samples. We gave 1 second of running time to our heuristic, a restricted time that would allow its use in a user-interactive service deployment process. We plot in Figure 4.4 the distance of clients in the network to its closest facility when selected with each of the explained methodologies.

Figure 4.5 shows the cost incurred by each of the service allocations obtained with the different methodologies.

As can be observed in Figure 4.4, our simulation-based methodology selects nodes in the network to host a single service that are closer to all other ones while maintaining a low deployment cost. Observing the disparate distances and costs obtained with both greedy methodologies, we can also deduct that (a) selecting those nodes with higher closeness centrality values results in low network distances and restrained total deployment costs; and (b) selecting nodes with lower closeness centrality results in very low opening costs but prohibitively high connecting ones and longer network distances. Both facts are a direct consequence of the cost function used to assign open costs for facilities in the FLP instance, but they help to value our proposal on finding low distance and low cost deployments.

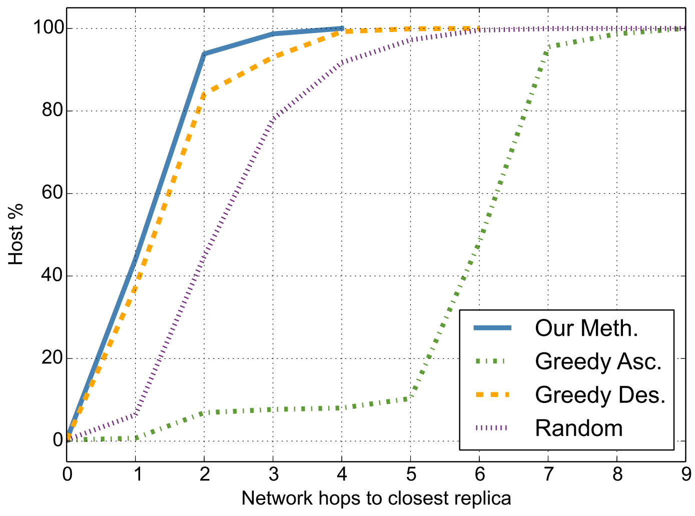

Figure 4.4: Percentage of clients at different distances to the closest replica

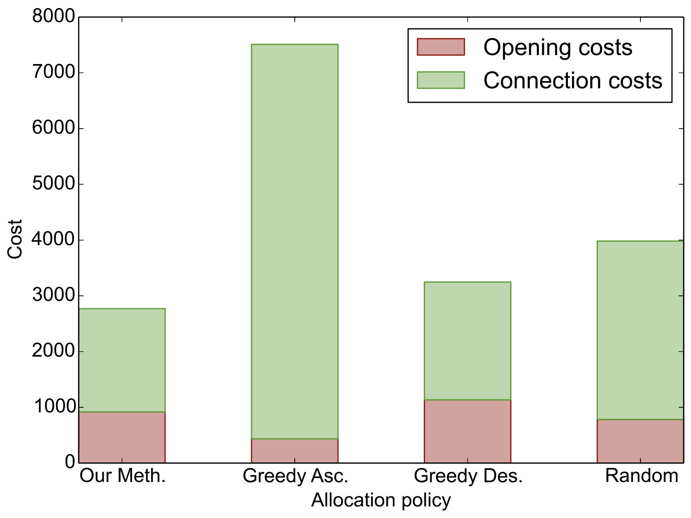

Figure 4.5: Service allocation costs comparison of the different deployments obtained by the above described methodologies

We already stated that in a real-life scenario several services or applications should be concurrently supported in the same network. To show the behavior of our methodology in such case, we evaluated the network distances after five consecutive service allocations, considering only one service replica was supported at a time on each host. We show in Figure 4.6 the distance of each host in the network to the closest replica of each of the five deployed services when they are allocated with the earlier explained methodologies.

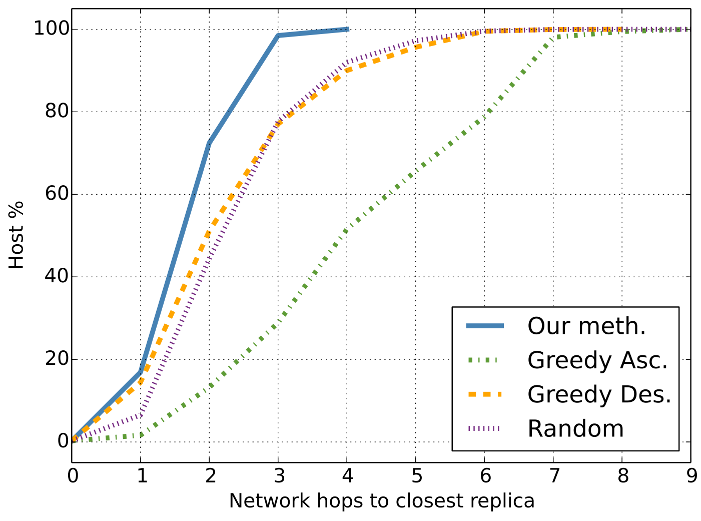

Figure 4.6: Percentage of clients at different distances to the closest replica

Figure 4.7 shows the aggregated costs for 5 services incurred by the allocations found with all the compared methodologies.

Figure 4.6 indicates our methodology consistently allocates replicas closer to all clients in the network by using the same number of resources. Specially, if we look at the maximum distance, our proposal is able to allocate replicas at half the distance than other approaches. Thus, we can highlight our methodology does a better resource utilization, getting lower network distances without increasing the number of consumed resources.

Moreover, as the bar graph in Figure 4.7 reflects, the selected service allocations are also cheaper on the aggregate cost after five consecutive allocations. According to the selected setup costs for the hosts, this fact suggests our methodology is able to share out the most suitable network spots (in terms of distance to all clients) among different service allocations. In a real deployment, this fact would help to improve the network allocation of several concurrent services.

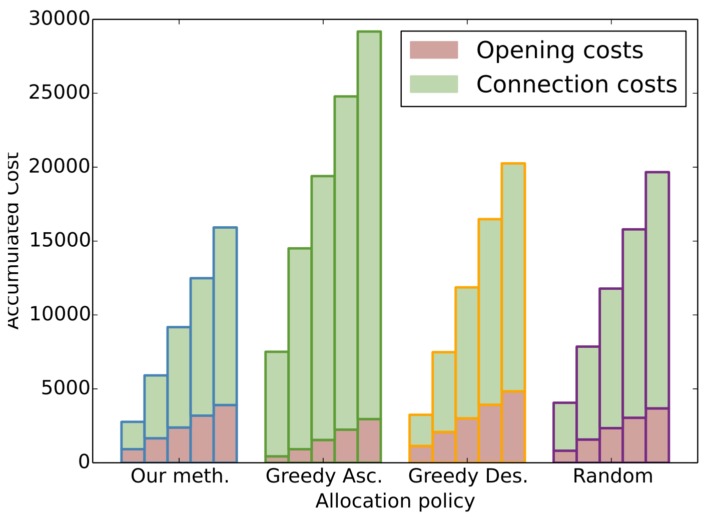

Figure 4.7: Accumulated costs for each of the five deployments with each methodology

4.7 Summary

Commodity network connections encouraged the appearance of volunteer and contributory computing. Despite the increasing quality of domestic Internet connections, host geographical spread may cause large network distances among participant hosts. In this chapter we intended to select the proper hosts in such a way network distances from all clients in a network to a set of contributory service deployed across a topology were minimized.

We modeled this problem as a Facility Location Problem, a classical optimization problem deeply studied in the fields of logistics and transportation. To address it, we proposed a methodology based on an Iterated Local Search framework to determine a subset of nodes in a topology to support a given service.

Our method proved to be competitive with current stat-of-the art heuristics in the well-established benchmarks for the FLP, specially in small- and medium-sized instances of the problem.

We then applied our method to a realistic scenario. We modeled the topology of a community network into an FLP and tested the deployment of several services on the available hosts. Our method discovered closer and cheaper deployments than the other methods we compared it to.

5 Conclusions and future work

5.1 Final conclusions

The contributory computing model is still at its early stages, as many issues described in their definition [27] remain still unsolved. All the contributions of this thesis coincide on the fact their outcomes aim to be be a system improvement to encourage the adoption of this model. Yet, many other problems should be addressed to finally obtain reliable and attractive systems able to replace or complement current clouds.

We explained the importance of availability on systems aiming to offer long-lived services on top of non-dedicated resources. Users will not be willing to move the services they currently hire to cloud vendors if the alternative is not reliable enough to offer long-lived services. We proposed an availability-aware method for services created from several connected components. Our proposal showed an efficient way to select those hosts in a community that could guarantee a certain level of service availability over time while minimizing a cost function associated to the utilization of each host. This research was published in the journal *Simulation: Transactions of the Society for Modeling and Simulation International* [P3].

Service availability was also the main concern behind the design process of our distributed social network, HorNet. This application was meant from the beginning to be deployed as a contributory service, so it helped us to motivate the goodness of the contributory model and its validation reinforced our point of view about the utility of user-donated computers. Moreover, we could apply our availability-aware allocation method for composite services in an actual application. With that, proper nodes to host HorNet services were selected and a given service availability level could be guaranteed. The HorNet design and validation was published in the journal *IEEE Internet Computing* [P1].

Our study over the *green computing* arena made us realize the tremendous efforts in different disciplines devoted to minimize the energy consumption of computing systems. Although the wide spectrum of various methods, we proposed a scheduling heuristic for ordered-task execution that could minimize the employed energy by minimizing makespans; and a simple and fast heuristic that combined our availability-aware methodology with the energy consumption of hosts in contributory platforms. In both cases, our proposals showed to be valuable by attaining significant energy consumption reductions and they were published in the *Journal of Applied Operational Research* [P2] and in the proceedings of the *Conference of* *the Spanish Association for Artificial Intelligence* [P4] and the *Winter Simulation Conference 2013* [P5].

The florescence of Community Networks brought many new parameters to take into account when selecting hosts to allocate service replicas. Their monitoring systems and the community commitment to publicly offer information about nodes, links and network topologies opened the door to use this information to manage resources more efficiently. As an example of a fruitful usage of this data, we focused on selecting replicas as close as possible to service consumers, proposing a methodology that minimizes network distances while did not favored more central nodes in the network. This research was accepted for publication in the proceedings of the *Winter Simulation Conference 2014* [P6].

Most of the proposed methods in this dissertation were inspired in classical optimization problems. These problems have been deeply studied in other research disciplines, such as logistics, transportation or manufacturing. We showed along this thesis that the influence from these already-existing tools to the distributed computing research field leads to novel and helpful methodologies to obtain affordable system optimizations.

5.2 Directions for future work

Seizing non-dedicated computing resources poses many research challenges. As part of this thesis, we partly addressed some of these challenges, but many of them still remain unexplored. Among them, the following ones are of particular interest from our point of view.

- We studied the application of computer simulation on the estimation of composite services availability, but we only considered predefined topologies. With all, simulation tools have proven to be extremely valuable to detect weaknesses on complex system topologies. Obtaining knowledge from components in a service topology, the service could be reinforced by modifying the topology adding redundancies or by assigning better hosts to these components.

- The methodologies presented in this thesis were validated by means of system simulation, mostly with artificially generated data. It would be of great interest to validate such methods in a real contributory platform. Our proposals could be easily integrated into the host selection step of current or future implementations of the paradigm.

- In the latter scenario, it would be extremely valuable to observe and study the resource utilization, in such a way new strategies could be introduced to improve allocations.

- Contributory platforms aggregate very heterogeneous nodes with some traits that can change and evolve over time. Our proposals focused on evaluating the allocations at the time of the host selection, but no transient study was performed over the systems. It would be relevant to determine how far our assignations would be valid in a real system, so re-assignation policies could be designed in order to make the services last for longer periods.

- Network-aware allocation mechanisms are a profitable tool that should be further exploded. We focused on minimizing network distances from replicas to clients, but many other helpful strategies could be adopted. Minimizing service overlay diameter, maximize available bandwidth among replicas or minimize delays from replicas to clients are just a small portion of the numerous possibilities offered to upper-layer services to take advantage from the information about the underlying network.

Bibliography

[1] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers, Inc., San Francisco, USA, 1999.

[2] P. Huber and M. Mills. Dig more coal–the pcs are coming. *Forbes*, 163(11):70–72, 1999.

[3] R. Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pages 101–102. IEEE, 2001.

[4] D.P. Anderson. Boinc: A system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4–10. IEEE, 2004.

[5] F. Xhafa and A. Abraham. Computational models and heuristic methods for grid scheduling problems. *Future generation computer systems*, 26(4):608–621, 2010.

[6] L. Pamies-Juarez, P. García-López, M. Sánchez-Artigas, and B. Herrera. Towards the design of optimal data redundancy schemes for heterogeneous cloud storage infrastructures. *Computer Networks*, 55(5):1100–1113, 2011.

[7] J. Kołodziej and F. Xhafa. Integration of task abortion and security requirements in ga-based meta-heuristics for independent batch grid scheduling. *Computers & Mathematics with Applications*, 63(2):350–364, 2012.

[8] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed systems: concepts and design.* pearson education, 2005.

[9] I. Foster. What is the grid? a three point checklist. grid today 1 (6). *URL http://www. gridtoday. com/02/0722/100136. html*, 2002.

[10] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International journal of high performance computing applications*, 15(3):200–222, 2001.

[11] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. Ieee, 2008.

[12] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53:50–58, April 2010.

[13] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica. Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28:13, 2009.

[14] P. Hofmann and D. Woods. Cloud computing: the limits of public clouds for business applications. *Internet Computing, IEEE*, 14(6):90–93, 2010.

[15] R. Rodrigues and P. Druschel. Peer-to-peer systems. *Communications of the ACM*, 53(10):72–82, 2010.

[16] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001*, pages 329–350. Springer, 2001.

[17] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM Computer Communication Review*, volume 31, pages 149–160. ACM, 2001.

[18] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. *Computer Science*, 2001.

[19] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In *Peer-to-Peer Systems IV*, pages 205–216. Springer, 2005.

[20] Murder: Large scale server deploys using BitTorrent and the BitTornado library. `http://github.com/lg/murder`.

[21] Facebook uses BitTorrent, and they love it, 2010. `http://torrentfreak.com/facebook-uses-bittorrent-and-they-love-it-100625/`.

[22] X. Zhang, J. Liu, B. Li, and T. Yum. Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 2102–2111. IEEE, 2005.

[23] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Consulted*, 1:2012, 2008.

[24] D.P. Anderson and G. Fedak. The computational and storage potential of volunteer computing. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 1, pages 73–80. IEEE, 2006.

[25] M. Litzkow, M. Livny, and M. Mutka. Condor-a hunter of idle workstations. In *Distributed Computing Systems, 1988., 8th International Conference on*, pages 104–111. IEEE, 1988.

[26] A. Chien, B. Calder, S. Elbert, and K. Bhatia. Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing*, 63(5):597–610, 2003.

[27] D. Lázaro. *A Middleware for Service Deployment in Contributory Computing Systems.* PhD thesis, Universitat Oberta de Catalunya, July 2011.

[28] N. Gold, A. Mohan, C. Knight, and M. Munro. Understanding service-oriented software. *Software, IEEE*, 21(2):71–77, 2004.

[29] M. Bell. *Service-oriented modeling (SOA): Service analysis, design, and architecture.* John Wiley & Sons, 2008.

[30] D. Lázaro, J.M. Marquès, and J. Jorba. Towards an architecture for service deployment in contributory communities. *International Journal of Grid and Utility Computing*, 1(3):227–238, 2009.

[31] D. Lázaro, J.M. Marquès, and X. Vilajosana. Flexible resource discovery for decentralized p2p and volunteer computing systems. In *Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2010 19th IEEE International Workshop on*, pages 235–240. IEEE, 2010.

[32] D. Lázaro, D. Kondo, and J.M. Marquès. Long-term availability prediction for groups of volunteer resources. *Journal of Parallel and Distributed Computing*, 72(2):281–296, 2012.

[33] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, 2003.

[34] G. Briscoe and A. Marinos. Digital ecosystems in the clouds: Towards community cloud computing. In *Digital Ecosystems and Technologies, 2009. DEST'09. 3rd IEEE International Conference on*, pages 103–108. IEEE, 2009.

[35] A. Marinos and G. Briscoe. Community cloud computing. In *Proceedings of the 1st International Conference on Cloud Computing*, CloudCom '09, pages 472–484, Berlin, Heidelberg, 2009. Springer-Verlag.

[36] A. Chandra and J. Weissman. Nebulas: using distributed voluntary resources to build clouds. In *Proceedings of the 2009 conference on Hot topics in cloud computing*, Hot-Cloud'09, pages 2–2, Berkeley, CA, USA, 2009. USENIX Association.

[37] J.B. Weissman, P. Sundarrajan, A. Gupta, M. Ryden, R. Nair, and A. Chandra. Early experience with the distributed nebula cloud. In *Proceedings of the fourth international workshop on Data-intensive distributed computing*, pages 17–26. ACM, 2011.

[38] M. Ryden, A. Chandra, and J. Weissman. Nebula: Data intensive computing over widely distributed voluntary resources. Technical report, Citeseer, 2013.

[39] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[40] B. Yee, D. Sehr, G. Dardyk, J. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, and N. Fullagar. Native client: A sandbox for portable, untrusted x86 native code. In *Security and Privacy, 2009 30th IEEE Symposium on*, pages 79–93. IEEE, 2009.

[41] F. Costa, Luis S., and M. Dahlin. Volunteer cloud computing: Mapreduce over the internet. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 1855–1862. IEEE, 2011.

[42] V. Cunsolo, S. Distefano, A. Puliafito, and M. Scarpa. Volunteer computing and desktop cloud: The cloud@ home paradigm. In *Network Computing and Applications, 2009. NCA 2009. Eighth IEEE International Symposium on*, pages 134–139. IEEE, 2009.

[43] V. Cunsolo, S. Distefano, A. Puliafito, and M. Scarpa. Cloud@home: bridging the gap between volunteer and cloud computing. In *Proceedings of the 5th international conference on Emerging intelligent computing technology and applications*, ICIC'09, pages 423–432, Berlin, Heidelberg, 2009. Springer-Verlag.

[44] R. Flickenger. *Building wireless community networks.* O'reilly, 2002.

[45] D. Fernández-Baca. Allocating modules to processors in a distributed system. *Software Engineering, IEEE Transactions on*, 15(11):1427–1436, 1989.

[46] E. Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.

[47] C. Lo and K. Qian. Green computing methodology for next generation computing scientists. In *Computer Software and Applications Conference (COMPSAC), 2010 IEEE 34th Annual*, pages 250–251. IEEE, 2010.

[48] A. Juan and M. Rabe. Combining simulation with heuristics to solve stochastic routing and scheduling problems. In *Proceedings of the 15th ASIM Dedicated Conference*, pages 641–650, 2013.

[49] H. Lourenço, O. Martin, and T. Stützle. *Iterated local search.* Springer, 2003.

[50] R. Ruiz and C. Maroto. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(2):479–494, 2005.

[51] M. Balinski. On finding integer solutions to linear programs. Technical report, DTIC Document, 1964.

[52] J. Stollsteimer. *The effect of technical change and output expansion on the optimum number, size, and location of pear marketing facilities in a California pear producing region.* PhD thesis, University of California, Berkeley, 1961.

[53] H. Pham. *Handbook of reliability engineering.* Springer Verlag, 2003.

[54] R. Bhagwan, S. Savage, and G. Voelker. Understanding Availability. In *Proceedings of IPTPS'03*, 2003.

[55] J.R. Douceur. Is remote host availability governed by a universal law? *SIGMETRICS Performance Evaluation Review*, 31(3):25–29, 2003.

[56] J. Faulin, A. Juan, S. Martorell, and J. Ramírez-Márquez. *Simulation Methods for Reliability and Availability of Complex Systems.* Springer, 2010.

[57] T. Bajenescu. Predict the reliability of complex systems by applying the Monte Carlo method. In *Optimization of Electrical and Electronic Equipments, 1998. OPTIM'98. Proceedings of the 6th International Conference on*, volume 3, pages 789–792. IEEE, 2002.

[58] A. Caprara, H. Kellerer, U. Pferschy, and D. Pisinger. Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operational Research*, 123:333–345, 2000.

[59] M. Mastrolilli and M. Hutter. Hybrid rounding techniques for knapsack problems. *Discrete Applied Mathematics*, 154:640–649, 2006.

[60] P. Triantafillou. High availability is not enough [distributed systems]. In *Management of Replicated Data, 1992., Second Workshop on the*, pages 40–43. IEEE, 1992.

[61] S. Hariri and H. Mutlu. Hierarchical modeling of availability in distributed systems. *Software Engineering, IEEE Transactions on*, 21(1):50–56, 1995.

[62] D. Long, A. Muir, and R. Golding. A Longitudinal Survey of Internet Host Reliability. In *14th Symposium on Reliable Distributed Systems*, pages 2–9, 1995.

[63] S. Saroiu, P.K. Gummadi, and S.D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedinsg of MMCN*, January 2002.

[64] J. Chu, K. Labonte, and B. Levine. Availability and locality measurements of peer-to-peer file systems. In *Proceedings of ITCom: Scalability and Traffic Control in IP Networks*, July 2003.

[65] W. Bolosky, J. Douceur, D. Ely, and M. Theimer. Feasibility of a Serverless Distributed file System Deployed on an Existing Set of Desktop PCs. In *Proceedings of SIGMETRICS*, 2000.

[66] D. Kondo, G. Fedak, F. Cappello, A. Chien, and H. Casanova. Characterizing resource availability in enterprise desktop grids. *Future Generation Comp. Syst.*, 23(7):888–903, 2007.

[67] S. Giesecke, T. Warns, and W. Hasselbring. Availability simulation of peer-to-peer architectural styles. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–6, 2005.

[68] J.W. Mickens and B.D. Noble. Exploiting availability prediction in distributed systems. In *Proceedings of the 3rd conference on Networked Systems Design & Implementation-Volume 3*, page 6. USENIX Association, 2006.

[69] A. Andrzejak, D. Kondo, and D.P. Anderson. Ensuring collective availability in volatile resource pools via forecasting. In *DSOM*, pages 149–161, 2008.

[70] A.G. Dimakis, P.B. Godfrey, Yunnan W., M.J. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. *Information Theory, IEEE Transactions on*, 56(9):4539 –4551, September 2010.

[71] R. Bhagwan, K. Tati, Y.C. Cheng, S. Savage, and G.M. Voelker. Total recall: System support for automated availability management. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 25–25, 2004.

[72] P. Knežević, A.s Wombacher, and T. Risse. Dht-based self-adapting replication protocol for achieving high data availability. In Ernesto Damiani, Kokou Yetongnon, Richard Chbeir, and Albert Dipanda, editors, *Advanced Internet Based Systems and Applications*, pages 201–210. Springer-Verlag, Berlin, Heidelberg, 2009.

[73] A. Andrzejak, D. Kondo, and D.P. Anderson. Exploiting non-dedicated resources for cloud computing. In *Network Operations and Management Symposium (NOMS), 2010 IEEE*, pages 341 –348, April 2010.

[74] Y. Dai, M. Xie, and K. Poh. Availability modeling and cost optimization for the grid resource management system. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 38(1):170 –179, jan. 2008.

[75] D. Kondo, A. Andrzejak, and D.P. Anderson. On correlated availability in internet-distributed systems. In *Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing*, pages 276–283. IEEE Computer Society, 2008.

[76] D. Kondo, B. Javadi, A. Iosup, and D. Epema. The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 398 –407, May 2010.

[77] B. Javadi, D. Kondo, J. Vincent, and D.P. Anderson. Discovering statistical models of availability in large distributed systems: An empirical study of seti@home. *IEEE Transactions on Parallel and Distributed Systems*, 22:1896–1903, 2011.

[78] R. Morales and I. Gupta. AVMON: Optimal and scalable discovery of consistent availability monitoring overlays for distributed systems. *Parallel and Distributed Systems, IEEE Transactions on*, 20(4):446–459, 2009.

[79] G. Attiya and Y. Hamam. Task allocation for maximizing reliability of distributed systems: a simulated annealing approach. *Journal of Parallel and Distributed Computing*, 66:1259–1266, October 2006.

[80] Q. Kang, H. He, H. Song, and R. Deng. Task allocation for maximizing reliability of distributed computing systems using honeybee mating optimization. *J. Syst. Softw.*, 83:2165–2174, November 2010.

[81] Y. Tao, H. Jin, and X. Shi. Grid workflow scheduling based on reliability cost. In *Proceedings of the 2nd international conference on Scalable information systems*, InfoScale '07, pages 12:1–12:8. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007.

[82] P.Y. Yin, S.S. Yu, P.P. Wang, and Y.T. Wang. A hybrid particle swarm optimization algorithm for optimal task assignment in distributed systems. *Computer Standards & Interfaces*, 28(4):441–450, 2006.

[83] P. Yin, S. Yu, P. Wang, and Y. Wang. Task allocation for maximizing reliability of a distributed system using hybrid particle swarm optimization. *J. Syst. Softw.*, 80:724–735, May 2007.

[84] P. Yin, S. Yu, P. Wang, and Y. Wang. Multi-objective task allocation in distributed computing systems by hybrid particle swarm optimization. *Applied Mathematics and Computation*, 184(2):407 – 420, 2007.

[85] X. Tang, K. Li, R. Li, and B. Veeravalli. Reliability-aware scheduling strategy for heterogeneous distributed computing systems. *J. Parallel Distrib. Comput.*, 70:941–952, September 2010.

[86] J. Yu, R. Buyya, and C.K. Tham. Cost-based scheduling of scientific workflow applications on utility grids. In *e-Science and Grid Computing, 2005. First International Conference on*, pages 8–pp. Ieee, 2005.

[87] L. Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. *Software Engineering, IEEE Transactions on*, 30(5):311–327, 2004.

[88] I. Brandic, S. Benkner, G. Engelbrecht, and R. Schmidt. Qos support for time-critical grid workflow applications. In *e-Science and Grid Computing, 2005. First International Conference on*, pages 8–pp. IEEE, 2005.

[89] J. Faulin, A. Juan, C. Serrat, and V. Bargueño. Predicting availability functions in time-dependent complex systems with saedes simulation algorithms. *Reliability Engineering & System Safety*, 93(11):1761–1771, 2008.

[90] J.M. Marquès, X. Vilajosana, T. Daradoumis, and L. Navarro. Lacolla: Middleware for self-sufficient online collaboration. *Internet Computing, IEEE*, 11(2):56–64, 2007.

[91] BBC News - Twitter to selectively 'censor' tweets by country, 2012. `http://www.bbc.co.uk/news/world-us-canada-16753729`.

[92] Decentralized online social networks. `http://sands.sce.ntu.edu.sg/dOSN/`.

[93] S. Seong, J. Seo, M. Nasielski, D. Sengupta, S. Hangal, S. Teh, R. Chu, B. Dodson, and M. Lam. Prpl: a decentralized social networking infrastructure. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, MCS '10, pages 8:1–8:8, New York, NY, USA, 2010. ACM.

[94] A. Shakimov, A. Varshavsky, L. Cox, and R. Cáceres. Privacy, cost, and availability tradeoffs in decentralized osns. In *Proceedings of the 2nd ACM workshop on Online social networks*, WOSN '09, pages 13–18, New York, NY, USA, 2009. ACM.

[95] C. Yeung, I. Liccardi, K. Lu, O. Seneviratne, and T. Berners-Lee. Decentralization: The future of online social networking.

[96] Diaspora project. https://joindiaspora.com/.

[97] S. Buchegger, D. Schiöberg, L. Vu, and A. Datta. Peerson: P2p social networking: early experiences and insights. In *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, SNS '09, pages 46–52, New York, NY, USA, 2009. ACM.

[98] O. Bodriagov and S. Buchegger. Encryption for p2p social networks. In *SPSN 2011, Workshop on Security and Privacy of Social Networks, in conjunction with IEEE SocialCom*, October 2011.

[99] Barracuda Labs 2010 Annual Security Report, 2010. http://www.barracudalabs.com/downloads/2010EndyearSecurityReportFINAL.pdf.

[100] The apache cassandra project. http://cassandra.apache.org/.

[101] J. Glanz. Power, pollution and the internet. *The New York Times*, 22, 2012.

[102] European Commission et al. Carbon footprint: What it is and how to measure it. *Accessed on April*, 15:2009, 2007.

[103] P. Laplante and S. Murugesan. It for a greener planet. *IT Professional*, 13(1):0016–18, 2011.

[104] S. Garg, C. Yeo, A. Anandasivam, and R. Buyya. Environment-conscious scheduling of hpc applications on distributed cloud-oriented data centers. *Journal of Parallel and Distributed Computing*, 71(6):732–749, 2011.

[105] L. Zhang, K. Li, and Y. Zhang. Green task scheduling algorithms with speeds optimization on heterogeneous cloud servers. In *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, pages 76–80. IEEE Computer Society, 2010.

[106] C. Belady, A. Rawson, J. Pfleuger, and T. Cader. Green grid data center power efficiency metrics: Pue and dcie. Technical report, Technical report, Green Grid, 2008.

[107] M. Stansberry and J. Kudritzki. Uptime institute 2012 data center industry survey. *Uptime Institute Survey*, 2012.

[108] Y. Duy, T.and Sato and Y. Inoguchi. Performance evaluation of a green scheduling algorithm for energy savings in cloud computing. In *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8. IEEE, 2010.

[109] D. Borgetto, H. Casanova, G. Da Costa, and J. Pierson. Energy-aware service allocation. *Future Generation Computer Systems*, 28(5):769–779, 2012.

[110] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *NSDI*, volume 8, pages 337–350, 2008.

[111] K. Le, R. Bianchini, M. Martonosi, and T. Nguyen. Cost-and energy-aware load distribution across data centers. *Proceedings of HotPower*, pages 1–5, 2009.

[112] D. Careglio, G. Da Costa, R. Kat, A. Mendelson, J. Pierson, and Y. Sazeides. Hardware leverages for energy reduction in large scale distributed systems. Technical report, Technical Report IRIT/RT-2010-2-FR, IRIT. University Paul Sabatier (Toulouse, May 2010), 2010.

[113] H. Yuan, C. Kuo, and I. Ahmad. Energy efficiency in data centers and cloud-based multimedia services: An overview and future directions. In *Green Computing Conference, 2010 International*, pages 375–382. IEEE, 2010.

[114] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M.Q. Dang, and K. Pentikousis. Energy-efficient cloud computing. *The Computer Journal*, 53(7):1045–1051, 2010.

[115] D. Kliazovich, P. Bouvry, and Samee U. Khan. Greencloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing*, 62(3):1263–1283, 2012.

[116] Y. Lee and A. Zomaya. Energy efficient utilization of resources in cloud computing systems. *The Journal of Supercomputing*, 60(2):268–280, 2012.

[117] A. Beloglazov, J. Abawajy, and R. Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28(5):755–768, 2012.

[118] V. Valancius, N. Laoutaris, L. Massoulié, C. Diot, and P. Rodriguez. Greening the internet with nano data centers. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 37–48. ACM, 2009.

[119] B. Schott and A. Emmen. Green methodologies in desktop-grid. In *Computer Science and Information Technology (IMCSIT), Proceedings of the 2010 International Multiconference on*, pages 671–676. IEEE, 2010.

[120] B. Schott and A. Emmen. Green desktop-grids: scientific impact, carbon footprint, power usage efficiency. *Scalable Computing: Practice and Experience*, 12(2), 2011.

[121] R. Harmon and N. Auseklis. Sustainable it services: Assessing the impact of green computing practices. In *Management of Engineering & Technology, 2009. PICMET 2009. Portland International Conference on*, pages 1707–1717. IEEE, 2009.

[122] R. Harmon and H. Demirkan. The next wave of sustainable it. *IT professional*, 13(1):19–25, 2011.

[123] S. Murugesan. Making it green. *IT professional*, 12(2):0004–5, 2010.

[124] A. Juan, B. Barrios, S. González-Martín, J. Faulin, M. Coccola, and T. Bektas. Combining biased randomization with meta-heuristics for solving the multi-depot vehicle routing problem. In *Proceedings of the Winter Simulation Conference*, page 347. Winter Simulation Conference, 2012.

[125] A. Juan, J. Faulin, J. Jorba, J. Caceres, and J.M. Marquès. Using parallel & distributed computing for real-time solving of vehicle routing problems with stochastic demands. *Annals of Operations Research*, 207(1):43–65, 2013.

[126] M. Pinedo. Minimizing the expected makespan in stochastic flow shops. *Operations Research*, 30(1):148–162, 1982.

[127] B. Dodin. Determining the optimal sequences and the distributional properties of their completion times in stochastic flow shops. *Computers & operations research*, 23(9):829–843, 1996.

[128] M. Gourgand, N. Grangeon, and S. Norre. A contribution to the stochastic flow shop scheduling problem. *European Journal of Operational Research*, 151(2):415–433, 2003.

[129] M. Gourgand, N. Grangeon, and S. Norre. Markovian analysis for performance evaluation and scheduling in¡ i¿ m¡/i¿ machine stochastic flow-shop with buffers of any capacity. *European Journal of Operational Research*, 161(1):126–147, 2005.

[130] L. Wang, L. Zhang, and D. Zheng. Genetic ordinal optimisation for stochastic flow shop scheduling. *The International Journal of Advanced Manufacturing Technology*, 27(1-2):166–173, 2005.

[131] L. Wang, L. Zhang, and D. Zheng. A class of hypothesis-test-based genetic algorithms for flow shop scheduling with stochastic processing time. *The International Journal of Advanced Manufacturing Technology*, 25(11-12):1157–1163, 2005.

[132] K. Baker and D. Trietsch. Three heuristic procedures for the stochastic, two-machine flow shop problem. *Journal of Scheduling*, 14(5):445–454, 2011.

[133] H. Allaoui, S. Lamouri, and M. Lebbar. A robustness framework for a stochastic hybrild flow shop to minimize the makespan. In *Service Systems and Service Management, 2006 International Conference on*, volume 2, pages 1097–1102. IEEE, 2006.

[134] S.H. Choi and K. Wang. Flexible flow shop scheduling with stochastic processing times: A decomposition-based approach. *Computers & Industrial Engineering*, 63(2):362–373, 2012.

[135] K. Kianfar, S. Fatemi Ghomi, and A. Oroojlooy Jadid. Study of stochastic sequence-dependent flexible flow shop via developing a dispatching rule and a hybrid ga. *Engineering Applications of Artificial Intelligence*, 25(3):494–506, 2012.

[136] Q. Zhou and X. Cui. Research on multiobjective flow shop scheduling with stochastic processing times and machine breakdowns. In *Service Operations and Logistics, and Informatics, 2008. IEEE/SOLI 2008. IEEE International Conference on*, volume 2, pages 1718–1724. IEEE, 2008.

[137] K.R. Baker and D. Altheimer. Heuristic solution methods for the stochastic flow shop problem. *European Journal of Operational Research*, 216(1):172–177, 2012.

[138] H. Campbell, R. Dudek, and M. Smith. A heuristic algorithm for the n job, m machine sequencing problem. *Management science*, 16(10):B–630, 1970.

[139] M. Nawaz, E. Enscore Jr, and I. Ham. A heuristic algorithm for the m-machine,n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983.

[140] R. Ruiz and T. Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, 2007.

[141] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

[142] S. Luke. *Essentials of metaheuristics*, volume 3. Lulu Raleigh, 2009.

[143] R. LaMothe. Edge computing. Technical report, Pacific Northwest National Laboratory, January 2013.

[144] Athena Vakali and George Pallis. Content delivery networks: Status and trends. *Internet Computing, IEEE*, 7(6):68–74, 2003.

[145] G. Cornuejols, M. L Fisher, and G. Nemhauser. Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management science*, 23(8):789–810, 1977.

[146] V. Verter. Uncapacitated and capacitated facility location problems. In *Foundations of Location Analysis*, pages 25–37. Springer, 2011.

[147] G. Cornuejols, G. Nemhauser, and L. Wolsey. The uncapacitated facility location problem. Technical report, DTIC Document, 1983.

[148] J. Vygen. *Approximation Algorithms Facility Location Problems.* Forschungsinstitut für Diskrete Mathematik, Rheinische Friedrich-Wilhelms-Universität, 2005.

[149] Z. Drezner. *Facility location: a survey of applications and methods.* Springer Verlag, 1995.

[150] L. Snyder. Facility location under uncertainty: a review. *IIE Transactions*, 38(7):547–564, 2006.

[151] D. Fotakis. Online and incremental algorithms for facility location. *ACM SIGACT News*, 42(1):97–131, 2011.

[152] M. Efroymson and T. Ray. A branch-bound algorithm for plant location. *Operations Research*, 14(3):361–368, 1966.

[153] K. Spielberg. Algorithms for the simple plant-location problem with some side conditions. *Operations Research*, 17(1):85–111, 1969.

[154] L. Schrage. Implicit representation of variable upper bounds in linear programming. In *Computational practice in mathematical programming*, pages 118–132. Springer, 1975.

[155] D. Erlenkotter. A dual-based procedure for uncapacitated facility location. *Operations Research*, 26(6):992–1009, 1978.

[156] M. Körkel. On the exact solution of large-scale simple plant location problems. *European Journal of Operational Research*, 39(2):157–173, 1989.

[157] D. Hochbaum. Heuristics for the fixed cost median problem. *Mathematical programming*, 22(1):148–162, 1982.

[158] D. Shmoys, É. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 265–274. ACM, 1997.

[159] F. Chudak. *Improved approximation algorithms for uncapacitated facility location.* Springer, 1998.

[160] K. Jain and V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 2–13. IEEE, 1999.

[161] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing lp. *Journal of the ACM (JACM)*, 50(6):795–824, 2003.

[162] S. Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Information and Computation*, 222:45–58, 2013.

[163] A. Kuehn and M. Hamburger. A heuristic program for locating warehouses. *Management science*, 9(4):643–666, 1963.

[164] M. Alves and M. Almeida. Simulated annealing algorithm for the simple plant location problem: A computational study. *Revista Investigacao Operacional*, 12, 1992.

[165] J. Kratica, D. Tošic, V. Filipović, and I. Ljubić. Solving the simple plant location problem by genetic algorithm. *RAIRO-Operations Research*, 35(01):127–142, 2001.

[166] D. Ghosh. Neighborhood search heuristics for the uncapacitated facility location problem. *European Journal of Operational Research*, 150(1):150–162, 2003.

[167] L. Michel and P. Van Hentenryck. A simple tabu search for warehouse location. *European Journal of Operational Research*, 157(3):576–591, 2004.

[168] M. Resende and R. Werneck. A hybrid multistart heuristic for the uncapacitated facility location problem. *European Journal of Operational Research*, 174(1):54–68, 2006.

[169] M. Lai, H. Sohn, T. Tseng, and C. Chiang. A hybrid algorithm for capacitated plant location problem. *Expert Systems with Applications*, 37(12):8599–8605, 2010.

[170] D. Wang, Q. Wang, Y. Yan, and H. Wang. An adaptive version of parallel mpso with openmp for uncapacitated facility location problem. In *Control and Decision Conference, 2008. CCDC 2008. Chinese*, pages 2387–2391. IEEE, 2008.

[171] L. Cooper. Location-allocation problems. *Operations Research*, 11(3):331–343, 1963.

[172] J. Klincewicz and H. Luss. A dual-based algorithm for multiproduct uncapacitated facility location. *Transportation Science*, 21(3):198–206, 1987.

[173] A. Meyerson. Online facility location. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 426–431. IEEE, 2001.

[174] E. Carrizosa, A. Ushakov, and I. Vasilyev. A computational study of a nonlinear minsum facility location problem. *Computers & Operations Research*, 39(11):2625–2633, 2012.

[175] V. Balachandran and S. Jain. Optimal facility location under random demand with general cost structure. *Naval Research Logistics Quarterly*, 23(3):421–436, 1976.

[176] D. Karget and M. Minkoff. Building steiner trees with incomplete global knowledge. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 613–623. IEEE, 2000.

[177] F. Thouin and M. Coates. Equipment allocation in video-on-demand network deployments. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, 5(1):5, 2008.

[178] G. Lee and A. Murray. Maximal covering with network survivability requirements in wireless mesh networks. *Computers, Environment and Urban Systems*, 34(1):49–57, 2010.

[179] M. Marić, Z. Stanimirović, and S. Božović. Hybrid metaheuristic method for determining locations for long-term health care facilities. *Annals of Operations Research*, pages 1–21, 2013.

[180] N. Brahimi and S. Khan. Warehouse location with production, inventory, and distribution decisions: a case study in the lube oil industry. *4OR*, pages 1–23, 2013.

[181] H. Lourenço, O. Martin, and T. Stützle. Iterated local search: Framework and applications. In *Handbook of Metaheuristics*, pages 363–397. Springer, 2010.

[182] A. Juan, J. Faulín, J. Jorba, D Riera, D. Masip, and B. Barrios. On the use of monte carlo simulation, cache and splitting techniques to improve the clarke and wright savings heuristics. *Journal of the Operational Research Society*, 62(6):1085–1097, 2011.

[183] S. González-Martín, A. Juan, D. Riera, Q. Castellà, R. Muñoz, and A. Pérez. Development and assessment of the sharp and randsharp algorithms for the arc routing problem. *AI Communications*, 25(2):173–189, 2012.

[184] G. Kant, M. Jacks, and C. Aantjes. Coca-cola enterprises optimizes vehicle routes for efficient product delivery. *Interfaces*, 38(1):40–50, 2008.

[185] M. Resende and C. Ribeiro. Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. In *Handbook of Metaheuristics*, pages 283–319. Springer, 2010.

[186] S. González, D. Riera, A. Juan, M. Elizondo, and P. Fonseca. Sim-randsharp: A hybrid algorithm for solving the arc routing problem with stochastic demands. In *Simulation Conference (WSC), Proceedings of the 2012 Winter*, pages 1–11. IEEE, 2012.

[187] A. Juan, H. Lourenço, M. Mateo, R. Luo, and Q. Castella. Using iterated local search for solving the flow-shop problem: parametrization, randomization and parallelization issues. *International Transactions in Operational Research*, 61(1):103–126, 2014.

[188] P. L'Ecuyer, L. Meliani, and J. Vaucher. Ssj: Ssj: a framework for stochastic simulation in java. In *Proceedings of the 34th conference on Winter simulation: exploring new frontiers*, pages 234–242. Winter Simulation Conference, 2002.

[189] M Hoefer. Ufllib, benchmark instances for the uncapacitated facility location problem. *URL: http://www. mpi-inf. mpg. de/departments/d1/projects/benchmarks/UflLib*, 2003.

[190] O. Bilde and J. Krarup. Sharp lower bounds and efficient algorithms for the simple plant location problem. *Annals of Discrete Mathematics*, 1:79–97, 1977.

[191] Y. Kochetov and D. Ivanenko. Computationally difficult instances for the uncapacitated facility location problem. In *Metaheuristics: Progress as Real Problem Solvers*, pages 351–367. Springer, 2005.

[192] S. Ahn, C. Cooper, G. Cornuejols, and A. Frieze. Probabilistic analysis of a relaxation for the k-median problem. *Mathematics of Operations Research*, 13(1):1–31, 1988.

[193] F. Barahona and F. Chudak. Near-optimal solutions to large-scale facility location problems. *Discrete Optimization*, 2(1):35–50, 2005.

[194] CONFINE Project, Community Networks Testbed for the Future Internet, 2011. `http://confine-project.eu/`.

[195] Clommunity Project, A Community networking Cloud in a box, 2013. `http://www.clommunity-project.eu/`.