Technical Report RT/37/2010

Flexible and Efficient Resource Location in Large-Scale Systems

João Alveirinho
INESC-ID/IST
jalveirinho@gsd.inesc-id.pt

João Paiva
INESC-ID/IST
jgpaiva@gsd.inesc-id.pt

João Leitão
INESC-ID/IST
jleitao@gsd.inesc-id.pt

Luis Rodrigues
INESC-ID/IST
ler@ist.utl.pt

Oct 2010

Abstract

Inherent to the cloud computing paradigm is the ability to adjust or deploy, on demand, services to face dynamic changes on the workload. Therefore, one needs an infrastructure to keep track of the resource allocation in the system and to efficiently locate the resources required to launch a new service, or re-dimension a running service. In this paper we propose a novel peer-to-peer architecture for efficient resource location that avoids the bottlenecks of centralized or hierarchical directory services.

Portions of this report have been published in the LADIS 2010: The 4th ACM SIGOPS/SIGACT Workshop on Large Scale Distributed Systems and Middleware.

Flexible and Efficient Resource Location in Large-Scale Systems

João Alveirinho João Paiva João Leitão Luís Rodrigues
INESC-ID/IST
{jalveirinho,jgpaiva,jleitao}@gsd.inesc-id.pt ler@ist.utl.pt

Abstract

Inherent to the cloud computing paradigm is the ability to adjust or deploy, on demand, services to face dynamic changes on the workload. Therefore, one needs an infrastructure to keep track of the resource allocation in the system and to efficiently locate the resources required to launch a new service, or re-dimension a running service. In this paper we propose a novel peer-to-peer architecture for efficient resource location that avoids the bottlenecks of centralized or hierarchical directory services.

1 Introduction

Inherent to the cloud computing paradigm is the ability to adjust or deploy, on demand, services that are able to face dynamic changes in the experienced load. Consider for example a sales web-service. In face of an increase in demand, one may be required to deploy the service over additional servers. The service may have specific requirements on the properties of the servers that can be used to deploy it, such as minimum CPU, number of cores, memory, disk space, etc. Furthermore, the location of the servers may also be a relevant selection criterion: if inter-server communication exists, one may attempt to allocate servers in the same network for efficiency reasons. Therefore, the challenge is to design techniques that support flexible and efficient resource location, in an environment that is composed by a very large, perhaps increasing, number of servers.

One simple solution to solve the resource location problem would be to use some form of centralized directory. For instance, a central server that would collect, and keep up-to-date, information concerning the resources available at each server. However, available resources are dynamic, as new services are deployed, old services are decommissioned, or services are relocated to increase efficiency or reduce power-consumption. As the number of servers increases, a central directory may easily become a bottleneck. Hierarchical directories mitigate this problem, but introduce additional complexity and traffic in order to maintain the structure. Furthermore, in order to achieve high availability, replication should be employed. However such mechanisms would significantly increase the bandwidth requirements of the system, an especially relevant factor when replicas are geographically distant. The challenges of keeping up-to-date information about a large-scale system have been discussed in several papers, for instance [15, 4].

In this paper we discuss an alternative solution that consists of using a novel peer-to-peer resource location strategy to support flexible and efficient resource location in the cloud. For that purpose we suggest a hybrid approach that combines unstructured and structured overlay networks, operating on distinct layers of an integrated architecture. All servers with available resources join an unstructured overlay network. This network self-organizes, using a low-cost background link biasing procedure: as a result, nodes with similar resources become neighbors in the unstructured overlay. Furthermore, some representative nodes of each region of the unstructured overlay join a distributed hash table (DHT) that is used to quickly route a query to the desired region of the overlay. Thus a query is first routed in the DHT to a node with some of the desired properties; then the vicinity of that node in the unstructured overlay is searched to find as many nodes as needed with the exact desired properties.

The rest of the paper is structured as follows. Section 2 presents our proposed architecture and describes the goals and operations of the more relevant components. Section 3 presents preliminary experimental results that validates our design. Section 4 discusses related work and finally, Section 5 concludes this paper discussing the applicability of our approach and presenting some directions for future work.

2 Architecture

In this section we describe the more relevant components of our proposal. The architecture of our solution combines an unstructured layer and a structured layer. In the following sections, we describe the main goals of each layer and how they operate. We then explain how these layers interact to route queries and support a dynamic resource allocation infrastructure.

Our architecture assumes that every resource available in the cloud, that can be allocated by clients, can be classified into some category from a set of categories defined a priori; we assume a finite number of categories, although this number can be arbitrarily high. For instance, we may classify nodes according to the amount of available memory as *memXS* (1.7Gb), *memS* (7.5Gb), *memM* (15 Gb), etc. Also, nodes can be classified according to the architecture in *32bits* or *64bits*. A similar approach can be used to classify other resources, such as disk space, number of cores, physical location, etc. A node is tagged with all the categories that characterize the resources it makes available.

When characterizing resources that can take a discrete value from a possibly large range, the definition of categories simplifies the task of defining which nodes are more similar. In these cases each category represents an interval I of values for that resource. Each peer in the system that shares that resource and has a value v for it, will fall into a category c so that $v \in I$.

For instance, if a node has 5Gb of available memory and a 32-bit processor, it would fall into the *memXS* category, since it has more than 1.7 Gb but less than 7.5Gb of available memory and, in terms of architecture, it would belong to the *32bits* category. The way these intervals are defined is application specific and could reflect the type of *virtual* machines that can be allocated in the cloud. Note that, a node with 5Gb of available memory can allocate a *virtual* machine that requires 1.7Gb (*memXS*) of memory but cannot be used to allocate a *virtual* machine that requires 7.5Gb (*memS*) of available memory.

2.1 Unstructured Layer

The purpose of the unstructured layer is to organize all nodes in the system (*i.e.* computational nodes of the cloud) that have available resources in a biased unstructured overlay network[1]. More specifically, we propose that peers run a self-organizing distributed algorithm, such as X-BOT [11], to adapt the overlay topology according to the resources available at each node.

X-BOT is a distributed protocol that is able to bias the topology of a symmetric unstructured overlay network (*i.e.* an overlay that denotes an undirected graph, such as the one proposed in [12]) given a proximity function that provides a measure of the "distance" between two nodes in the system, such that

[1]Note that, although our unstructured layer is biased by X-BOT, the resulting topology is still unstructured, as it does not follow any global and deterministic coordination scheme based on node identifiers (such as a DHT), and the topology remains random in nature.

nodes that are closer have higher probability of becoming neighbors. Naturally, the proximity function is application specific: for instance, it can reflect the roundtrip time between two nodes, or some higher level semantic relation. In our case, the proximity function reflects how similar the resources that two nodes have available are. In other words and as we detail further ahead in the text, we determine the distance between two nodes by taking into consideration the number of resource categories shared by those two nodes. X-BOT operates by iteratively applying an optimization procedure that involves the coordination of just 4 overlay nodes, and that swaps existing overlay links for "better" links. The protocol preserves the degree (*i.e.* the number of neighbors) of each node in the system during this process. Furthermore, it maintains a fixed number of random and unbiased "distant" links that prevent the emergence of excessive clustering in the network, preserving overlay connectivity by avoiding the creation of several disconnected clusters.

As stated before, X-BOT is used to bias the unstructured overlay topology such that each node becomes neighbor of other nodes that have similar available resources. For instance a node with categories *64bits* and *memM* is more likely to have other nodes with those categories as its neighbors than, say, a node with categories *32bits* and *memXS*. In a similar manner, among all nodes with categories *64bits* and *memM*, a node with category *datacenterA* is more likely to have another node with category *datacenterA* as a neighbor than a node with category *datacenterB*. The rationale for this strategy it to be able to efficiently process queries that search for multiple nodes with similar resources. Notice however, that each node in our system will still know other nodes with distinct profiles, as a result of X-BOT maintaining a set of "distant" links.

More precisely, our proximity metric when applied to a pair of nodes returns a distance value that depends on the number of available resource categories shared by those nodes. The returned value is zero if both nodes share exactly the same set of resource categories, and increases by a fixed amount for each category owned by only one of the nodes. Therefore, in our prototype X-BOT operates by exchanging overlay links, such that for each node n it maximizes the number of neighbors of n that share the same set of resource categories.

An important advantage of the unstructured layer is that it can operate with a very low and controlled cost: The biasing procedure induces a small amount of background traffic that can be adjusted by defining the period between consecutive optimization steps. As it happens with hierarchical structures, each change in nodes' resources involves message exchanges. However, whilst in a hierarchical system a small number of nodes may be overwhelmed with update messages, in our system this cost is shared amongst all neighbors of those nodes. Furthermore, since the overlay is unstructured, there are few constraints that need to be preserved, either when new nodes join the system or when the network has to be repaired due to node departures or failures.

2.2 Structured Layer

As we have noted above, the unstructured layer is able to create neighboring relations among nodes that have similar resources available. Therefore, when searching for resources, as soon as one finds a single node with the desired resources, one can expect to easily find other nodes with similar characteristics in the unstructured overlay vicinity. The purpose of the structured layer is to facilitate the first step of the resource location procedure.

For this purpose, a fraction of the nodes that belong to the unstructured overlay also join a DHT such as Chord [17] or Pastry [16]. The DHT is used to efficiently route a query to the desired region of the unstructured overlay. In the following, we describe how nodes coordinate to join the DHT.

The idea is that, in each region of radius r in the unstructured overlay, and for each category c present in that region, there is one node which is elected to represent c in the DHT. Such a node is called a *regional contact* for category c, or simply c-RC. A node that is a c-RC joins the DHT with an identifier constructed by assigning *hash(node_id)* to the s least significant bits and *hash(c)* to the remaining (most-significant) bits. This ensures that multiple contacts for the same category in different regions have different identifiers but are placed in a consecutive region of the DHT address space. Note that for popular resource categories, more than a single node can be elected to become a local representative of a given region, as long as they are not at r or least hops of distance from each other.

A c-RC node uses the unstructured overlay to periodically send a c-beacon to nodes in its r-vicinity. Other nodes that also own category c and receive the beacon, abstain from competing to become a c-RC. If a node that owns category c does not receive any beacon, it decides to compete with other potential candidates to become the c-regional contact. When multiple nodes compete, a simple bully election scheme is used to select which node becomes the c-RC. A node that is elected to be a c-RC, can use the leader of another category c' as a contact point for joining the DHT or, if there is none yet, it performs a random walk in the unstructured overlay to find a node that has information concerning *any* regional leader to use as a contact point to join the DHT.

Furthermore, each node may only be the c-regional contact for a single category. A node that already is a c-RC does not compete to become the regional contact for any of its other categories. Also, if a node is not a c-RC and detects that there are no regional contacts for multiple of its categories, it picks one of these categories at random and competes only for that category. Only if it loses an election, will it attempt to compete to become the regional contact for another category. This effectively distributes the routing load among the nodes of each region of r radius.

It is not impossible that a node becomes the only owner of two or more categories in a region. In this case, only one of these categories will be represented in the DHT. The resources owned by that node in those unique categories will not be found in queries that address exclusively those categories. However, these resources would still be found by queries that also target other resource categories in the region. This scenario is, however, extremely rare, as X-BOT effectively gathers nodes with the same category in the same region (the interested reader can refer to [11] for performance results of X-BOT using different distance metrics in distinct scenarios).

Let $\mathcal{C}$ be the set of all categories and N be the total number of nodes in the system. Consider also that each node maintains d neighbors in the biased unstructured overlay. We have determined experimentally that the effective number of nodes that belong to the DHT is low. In particular for the scenario used in our evaluation, for $\mathcal{C} = 17$, $d = 30$ and $r = 3$ in a system with 10.000 nodes only 53 nodes were required to join the structured overlay. Notice that this value is highly dependent on the parameter r. Therefore, assuming the existence of a distributed monitoring system, one could adjust the number of nodes that join the DHT in runtime, by manipulating the parameter r. This can be achieved by using an efficient gossip-based broadcast scheme, operating on top of the unstructured overlay network [12].

2.3 Query Routing

We now describe how to perform queries in our system. Our architecture does not constrain the format of the queries that can be performed. As queries are executed at each node, they can be arbitrarily complex, as long as it is possible to process them using the information available locally at each node. In order to route queries to nodes, there is only one requirement: from the query, it should be possible to extract the set $\mathcal{Q}$ of categories that match the query. For instance, assume that one is looking for a node with a 32 bit architecture, with $7.5Gb$ or $15Gb$ of memory, and located on data center A. In this particular case, we would need to extract the following set of matching categories: $\mathcal{Q} = \{$*memS*, *memM*, *32bits*, *datacenterA*$\}$.

The idea is to disseminate and process each query with some approximate message cost k. In our current prototype the value of k is static and defined offline. However, k could be dynamically adjusted to match the estimated rarity of the searched resource (for instance, based on the results returned by previous searches, for instance by leveraging on techniques similar to the ones proposed in [5]).

A query is disseminated as follows:

- First, the query is routed to the nearest member of the DHT (this is based on periodic beacons sent by regional contacts).
- Then a copy of the query is routed to each category $c \in \mathcal{Q}$ that matches the query in the DHT. Each copy will be received by a c-RC for that category. To promote load balancing, for each category c, the query is routed to an identifier composed by *hash(c)*$||\{s$ random bits$\}$. This ensures that different queries are injected into the unstructured layer via different representatives of category c in the DHT.
- Each c-RC starts a random walk of length $\frac{k}{|\mathcal{Q}|}$ in its vicinity. Random walks are biased to nodes that share more resource categories with the query.
- Each node visited by the random walks (including the c-RCs) executes the query locally and checks if it satisfies the query. In the affirmative case, it adds its own identifier to the query being disseminated in the random walk, and a flag that is set to true if it is a RC and to false otherwise.
- Finally, when a random walk reaches the maximum number of hops, it returns all matching nodes found to the source of the query.

Note that the total message cost of a query is k, plus the cost of reaching a DHT member from the source (typically 1), plus the number of hops in the DHT required to reach the c-RCs which is typically low, as it is in the order of $c \cdot \ln T$ where T is the number of nodes in the DHT. As we discussed previously, one can easily configure the system to promote a low value of T.

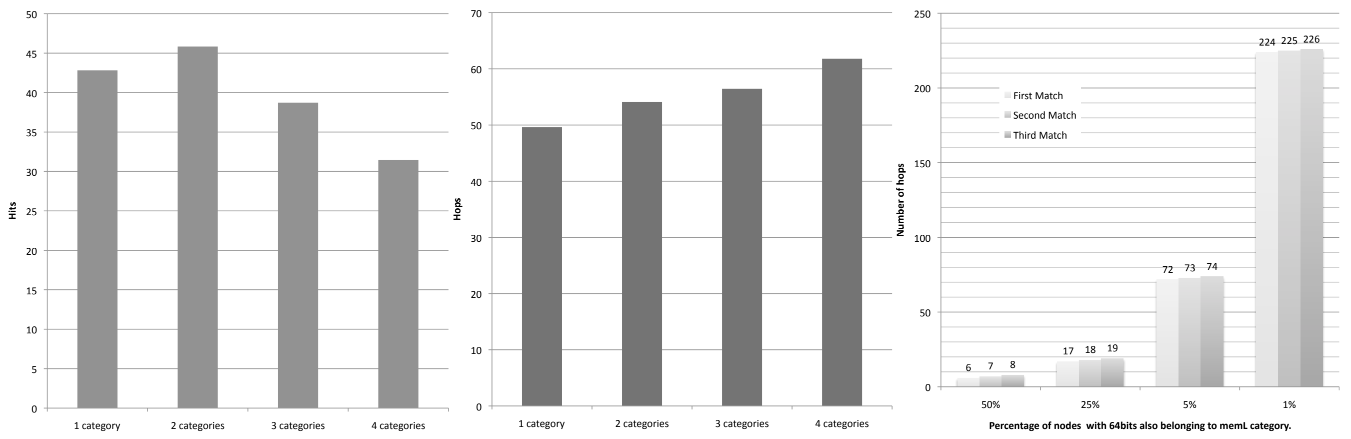
Because the unstructured overlay network topology is biased using the X-BOT protocol, the probability of these queries being forwarded to nodes that have the adequate amount of free resources to match the request is high. This allows us to efficiently and reliably support complex queries, without incurring in the typical overhead imposed by flooding solutions.

Also, we can slightly optimize the query dissemination process by having nodes avoiding to route the query to RCs of some categories, when the originator of the query already has some neighbors that match some of the categories in the set $\mathcal{Q}$. In this case, the originator of the query can use such neighbors as representative of these categories and leverage them to initiate the random walk in the unstructured overlay. This avoids the additional message cost of routing copies of the query, for those categories, through the DHT.

2.4 Resource Allocation

In response to a query, the source receives at most k nodes that match the query. Some of these nodes serve as regional contacts and others do not. Resources should be preferably allocated from nodes that are not RCs, as this improves the stability of the DHT. Only if not enough non-RC nodes are found, should resources be allocated from RCs.

Resource allocation itself is application dependent and orthogonal to our architecture. For instance, a new service may be deployed on the selected nodes or be expanded to use additional resources available in the cloud. When resources are effectively used on the nodes, the nodes update their categories accordingly.



(a) Number of Hits for Random Queries (b) Message Cost for Random Queries (c) Hops for Rare Intersections

Figure 1. Performance Results.

Note that when resources are allocated, the categories owned by a node may change. In response, X-BOT will iteratively move the node to another region of the unstructured overlay. However, our system does not require the unstructured overlay to have converged in order to satisfy queries. A node that does not mach a query simply forwards the random walk to one of its better suited neighbors.

3 Evaluation

In order to validate our approach, we now present some preliminary evaluation results that were obtained using the Peersim simulator [10] by employing its cycle-based engine. All results reported in this paper concern simulations performed over a system composed by 10.000 nodes.

Although most cloud computing providers do not disclose information on how the computing infrastructure is organized, we rely on a testing scenario based on the pre-configured instances provided by Amazon Web Services[1]. In our experimental setup, we have considered 17 distinct resource categories ($\mathcal{C} = 17$: 5 categories for different sizes of each resource type – CPU, Memory and Disk – and two categories for the CPU architecture – 32 and 64 bits), each node belongs to 4 of these categories (one per resource type) and maintains 30 unstructured overlay neighbors ($d = 30$). Finally, the radius to which nodes announce their presence in the DHT was set to 3 hops (therefore, $r = 3$). We consider that each physical host, in our simulated cloud computing infrastructure, owns twice the resources of the more powerful pre-configured virtual machine used by the Amazon Web Services.

We have evaluated our system for two types of queries. IIn this section we describe and motivate our experiments and present our preliminary results.

In the first experimental setting we have allocated approximately 20.000 virtual machines with random configurations (from the ones employed by Amazon) over the 10.000 physical nodes in order to promote the heterogeneity of available resources among nodes. We then execute 4.576 queries initiated from random nodes in the system, where we try to locate a machine with free resources in 1, 2, 3, and 4 different categories (we executed 1.144 queries of each type). We have set the k parameter (associated with the cost of performing random walks) to 48, therefore 48 nodes is the maximum number of nodes that can be returned by a query.

Our goal was to evaluate the number of nodes returned by each query (*i.e.* number of hits) and the cost, in number of messages, of disseminating queries targeting different number of categories. Results are reported in Figures 1(a) and 1(b) respectively. On average, queries are able to return a large set of answers. The number of nodes returned by queries mostly decreases with the increase of associated categories. This is expected, as a smaller number of nodes in the system have adequate resources to reply to such queries. However, even when performing queries for 4 resource categories, our approach is able to find more than 30 nodes that fit the requirements. Interestingly a query for a single resource category returns less elements that a query that targets 2 categories. We believe that this happens because, in such scenario only a single random walk is employed which can transverse, and exit, the overlay region biased for that category, being unable to locate additional valid answers afterwards.

In Figure 1(b) we report the message cost for disseminating each type of query. As expected the cost in number of messages slightly increases as the number of categories rises. This happens due to the additional cost of forwarding one message through the DHT for each query category. Notice however, that the cost does not rise linearly, as we maintain the cost of performing random walks constant ($k = 48$). Moreover, notice that, even when performing a query for 4 categories, the additional cost (*i.e.* the number of messages above 48) is, on average, below 15 additional messages.

We have also performed experiences in an additional setting where queries target two resource categories and each of these categories is individually very popular, but the intersection of both is rare. Notice that this is a scenario that presents a relevant challenge to our system, as the DHT will route queries to nodes belonging to each of the categories individually, starting random walks in regions of the unstructured overlay network that are biased for each category independently.

To this end we have configured our system to have half of the computational nodes belonging to category *64bits* (*i.e.* 5.000 nodes). We then varied the fraction of these nodes that also have the category *memL*. We then issued 5.000 individual queries for each tested configuration. We only routed a single query in the DHT to a representative of the *64bits* category, as this is the worst entry point for queries in this scenario. We measured the number of hops required to find to first, second, and third nodes that belong to both categories.

Results are depicted in Figure 1(c). As expected, as the percentage of nodes belonging to *64bits* that also belong to *memL* diminishes, the random walk requires additional hops to find the region in the unstructured overlay where nodes belonging to both categories are clustered. This happens because the queries are injected into the unstructured layer at representatives of a single category therefore it is possible that the region of the unstructured overlay where the query is injected is only biased for *64bits*. We believe however that such limitation can be easily circumvented, by allowing representatives of multiple categories to emerge. We will address this question as future work. Notice however that the biasing of the unstructured overlay network allows our system to find the second and third hit for the query in the hops immediately after locating the first hit.

4 Related Work

Managing resources in large-scale distributed infrastructures is not a new problem. A trivial solution is to rely on some centralized scheme, where a single node is responsible for maintaining information concerning the allocation and ownership of all resources available in the system. For instance, this solution is employed by the Condor [14] system. The central repository can then be directly queried by nodes that require additional resources. Because the repository maintains global knowledge on the system it can easily process complex queries. However, the overhead imposed on a single node, to maintain full and up-to-date information concerning all resources (and managing all resource allocation requests) is excessively high in a large-scale environment such as a cloud.

To address this issue the Globus toolkit [6] relies on multiple index servers, which replicate the information regarding the resource allocation among themselves. Although this removes the single point of failure, index servers are still required to maintain global up-to-date knowledge of the system, which might impose scalability limitations.

Alternatively, Eucalyptus [13] and Ovis [4] employ a hierarchy of aggregators that maintain resource allocation information for groups of nodes (clusters), and a top-level controller that is responsible for managing these cluster aggregators. Each of these aggregators is responsible for maintaining updated information concerning all resources of their cluster. Requests for additional resources, including those based on complex queries, can be directly processed by these aggregators. Both these systems require cluster aggregators to continually and actively monitor all nodes in their clusters, which incurs in significant overhead. Additionally the hierarchical structure adopted by these systems is not fault-tolerant.

DHTs such as Chord[17] or Pastry[16] can be used to maintain a distributed directory of existing resources in a scalable and fault-tolerant manner. Unfortunately, DHTs are specialized for performing exact match queries, which is extremely limitative in the cloud environment, where nodes might have heterogeneous profiles, and where cloud operators would benefit from minimizing the amount of unused resources in the cloud. Some DHT solutions have been proposed to address range queries [7, 2] however, these solutions are usually limited to operate on rages over a single dimension, or by limited query flexibility.

A typical way to support more complex queries in peer-to-peer systems is to rely in unstructured overlay networks and disseminate queries using some type of flooding technique. This is employed in systems such as Gnutella [18] or Gia [5]. This approach avoids the use of centralized repositories for maintaining information concerning available resources. However the use of flooding techniques imposes a significant overhead in nodes, especially if a large number of queries are performed simultaneously. Techniques exist to lower this overhead by employing limited or localized flooding, however these techniques sacrifice performance both in terms of recall rate (*e.g.* query success rate) and latency.

In [3], the authors explore a similar problem to the one addressed here. They propose a solution based on network slicing [9] to group nodes accordingly to their capacity in terms of a single and globally ordered metric. The authors argue that this can ease the resource allocation in the cloud. Unlike our work, this approach only takes into consideration a single type of resources whereas we organize the network taking into consideration an arbitrarily (large) number of resource categories, which allows us to process, in a more flexible way, more complex queries.

Finally, in [8] the authors study the use of random walks to perform resource location in a peer-to-peer grid. They conclude that in order to perform efficient and reliable resource location in large-scale environment, one requires more sophisticated approaches. Our work addresses these observations, as we rely both in a biased unstructured overlay network to provide locality properties to the overlay, and in an additional DHT to route queries to relevant areas of the unstructured overlay network.

5 Conclusions

We have proposed the initial design of a peer-to-peer architecture for efficient resource location for cloud and large-scale computing infrastructures, that avoids the bottlenecks of centralized or hierarchical directory services. Our proposal is based on a hybrid design and experimental results show that it is feasible, allowing the location of adequate resources in a large scale system, in a decentralized and efficient way.

As future work we plan to expand the design of our architecture, and explore ways to incorporate self-adaptive mechanisms to adapt the length of random walks used in the unstructured overlay network, and also to allow nodes to become representatives of multiple resource categories for relevant combinations.

Acknowledgments

This work was partially supported by FCT (INESC-ID multiannual funding) through the PIDDAC Program fund and by FCT projects "Redico" (PTDC/EIA/71752/2006) and "Pastramy" (PTDC/EIA/72405/2006). The authors wish to thank João Garcia and the anonymous reviewers for useful comments on this paper.

References

[1] Amazon web services. `http://aws.amazon.com/`.

[2] A. Andrzejak and Z. Xu. Scalable, efficient range queries for grid information services. In *In Proc. of the 2nd P2P'02*, page 33, Washington, DC, USA, 2002. IEEE Comp. Society.

[3] O. Babaoglu, M. Jelasity, A.-M. Kermarrec, A. Montresor, and M. van Steen. Managing clouds: a case for a fresh look at large unreliable dynamic networks. *SIGOPS Oper. Syst. Rev.*, 40(3):9–13, 2006.

[4] J. Brandt, A. Gentile, J. Mayo, P. Pebay, D. Roe, D. Thompson, and M. Wong. Resource monitoring and management with ovis to enable hpc in cloud computing environments. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1 –8, 23-29 2009.

[5] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *SIGCOMM*, pages 407–418, 2003.

[6] S. Fitzgerald. Grid information services for distributed resource sharing. In *HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing*, page 181, Washington, DC, USA, 2001. IEEE Computer Society.

[7] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: a scalable overlay network with practical locality properties. In *USITS'03: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*, page 9, Berkeley, CA, USA, 2003. USENIX Association.

[8] A. Iamnitchi, I. Foster, and D. C. Nurmi. A peer-to-peer approach to resource location in grid environments. In *HPDC '02: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, page 419, Washington, DC, USA, 2002. IEEE Computer Society.

[9] M. Jelasity and A.-M. Kermarrec. Ordered slicing of very large-scale overlay networks. In *P2P '06: Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing*, pages 117–124, Washington, DC, USA, 2006. IEEE Computer Society.

[10] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris. The Peersim simulator. `http://peersim.sf.net`.

[11] J. Leitão, J. P. Marques, J. Pereira, and L. Rodrigues. X-bot: A protocol for resilient optimization of unstructured overlays. In *Proceedings of the 28th IEEE International Symposium on Reliable Distributed Systems*, pages 236–245, Niagara Falls, New York, U.S.A., Sept. 2009.

[12] J. Leitão, J. Pereira, and L. Rodrigues. Hyparview: a membership protocol for reliable gossip-based broadcast. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 419–429, Edinburgh, UK, June 2007.

[13] S. Liu, Y. Liang, and M. Brooks. Eucalyptus: a web service-enabled e-infrastructure. In *CASCON '07: Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, pages 1–11, New York, NY, USA, 2007. ACM.

[14] R. Raman, M. Livny, and M. Solomon. Matchmaking: An extensible framework for distributed resource management. *Cluster Computing*, 2(2):129–138, 1999.

[15] R. V. Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21:2003, 2001.

[16] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.

[17] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM.

[18] D. Tsoumakos and N. Roussopoulos. Analysis and comparison of p2p search methods. In *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, page 25, New York, NY, USA, 2006. ACM.