Peer-to-peer networks for scalable grid landscapes in social agent simulations

B.G.W. Craenen*
B.Craenen@napier.ac.uk
B. Paechter*
B.Paechter@napier.ac.uk
*Centre for Emergent Computing
Napier University
10 Colinton Road
EH10 5DT, Edinburgh

Abstract

Recently, peer-to-peer networks have been proposed as the underlying architecture of large scale distributed social agent simulations. A number of problems arise when grid landscapes are used to represent the landscape in these simulations, primarily because, in a peer-to-peer network, the landscape has to be handled collectively by the nodes of the network. Two basic agent actions are identified as central to these problems: look and move. A solution to these problems is provided in which the network maintains a move-buffer and a look-index. Both solutions are configurable by the user of the simulation and provide a trade-off between the scalability of the system, the consistency of the information stored in the system, and the efficiency of the system.

1 Introduction

The size of the world that can be handled efficiently by a social agent simulation run on a single computer is restricted by the resources available on that computer. By combining the resources of several computers in a computer network, the efficient size of the world can be increased. These simulations are called distributed simulations. The architecture of the computer network that underlies a distributed simulation imposes restrictions on the efficiency of the simulation. In the NewTies[1] project, of which this paper is part, we propose to use a peer-to-peer (P2P) network as the underlying architecture for the simulation, since they impose fewer restrictions on the efficient size of the simulation. Social agent simulations commonly use a grid to represent the landscape on which the agents live. When an agent simulation is implemented on a single computer, a grid landscape is both straightforward and efficient to implement. In a distributed simulation this is not necessarily the case. Since a (pure) peer-to-peer network cannot have a central server, it has to partition the landscape so that it can be handled collectively by the nodes in the network. Because the peer nodes may differ in local configuration, processing speed, network bandwidth, and storage capacity, the size of the partitions can vary greatly, and can even dynamically change over time when new nodes become available and other nodes disappear. In this paper we discuss a solution to the problems that arise when a grid landscape has to be maintained in a peer-to-peer distributed social agent simulation.

This paper is organised as follows: section 2 discusses peer-to-peer networks in more detail. In section 3 we discuss how using a grid landscape in a peer-to-peer distributed social agent simulation necessitates the partitioning of the landscape over the nodes of the network and how this relates to the scalability of the network, the consistency of the information stored in the network, and the efficiency of the simulation. Two basic agent actions are identified as central to these issues: the move- and the look-action. In section 4 we discuss how the use of a *move-buffer* solves the problems imposed by the move-action. In section 5 we discuss how the use of a *look-index* solves the problems imposed by the look-action. The conclusions that can be draw from this paper are summarised in section 6.

[1] New and Emergent World-models Through Individual, Evolutionary, and Social Learning (NEW TIES), http://www.newties.org

2 Peer-to-peer networks

A peer-to-peer computer network is any network that does not rely on dedicated servers for communication but instead uses direct communication between clients (peers). A pure peer-to-peer network does not have the notion of clients or servers, but only peer nodes with equal functionality that simultaneously function as both clients and servers to the other nodes of the network.

The peer-to-peer network architecture differs from the client-server model in that in a client-server network, communication is usually relayed by the server, while in a peer-to-peer network, direct communication between the peers is the norm. A typical example for client-server communication is email, where the email is transmitted to the server for delivery, transmitted to the destination between servers, and is fetched later by the receiving client. In a client-server network, direct transmission from a client to another client is often impossible. Figure 1 shows a graphical representation of a simple client-server network with one server and four clients.

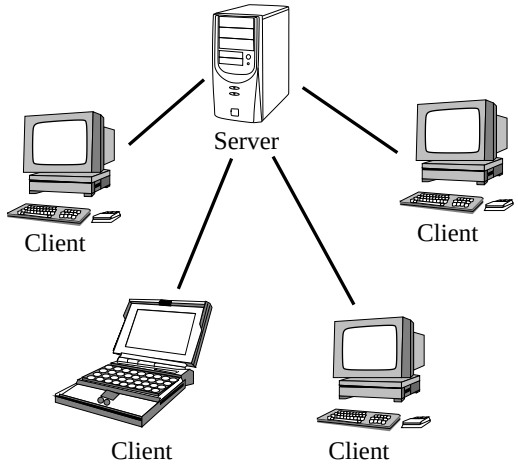

Figure 1: A client-server network

In a peer-to-peer network, any node is able to initiate or complete any supported transaction with any other node. Peer nodes may differ in local configuration, processing speed, network bandwidth, and storage quantity. A graphical representation of a simple peer-to-peer network is shown in figure 2.

There are two important properties of a peer-to-peer network that are of relevance here: the bandwidth of all peers can be fully used, and the network is able to maintain scalability when the number of nodes increases. This means that when the number of nodes in the network increases, the total available

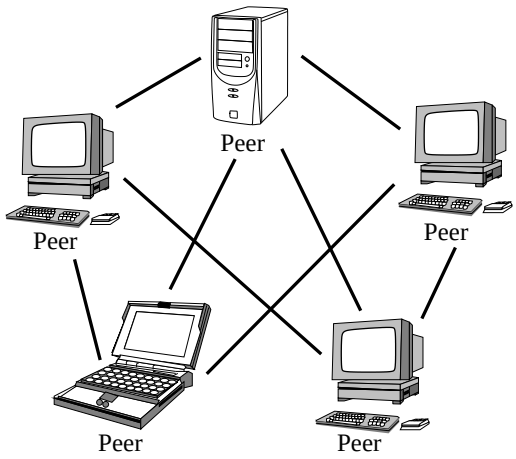

Figure 2: A peer-to-peer network

bandwidth also increases. In a client-server network, all clients have to share the (limited) bandwidth of the server, so that having a larger number of clients actually means slower data transfer.

Technically, a pure peer-to-peer application must implement only peer protocols that do not recognise the concepts of “server” and “client”. Such pure peer applications and networks are rare. Most networks and applications described as peer-to-peer actually contain or rely on some non-peer elements, such as the Domain Name System (DNS) of the Internet, which translates the IP-addresses of all computers connected to it into a human-readable hostname. Also, real-world peer-to-peer applications often use multiple protocols and act as client, server, and peer simultaneously, or over time. Many peer-to-peer systems use stronger peers (super-peers or super-nodes) as servers to which client-peers are connected in a star like fashion. The inherent scalability of peer-to-peer networks and their ability to collectively use all available bandwidth has attracted a great deal of attention to their application from computer science research. The advantages of peer-to-peer networks make them an interesting alternative underlying architecture for large scale distributed social agent simulations. The complexity of an social agent simulation increases with the scale of the simulation and for the simulation of some problems, large scale simulations are required. The resource need of these simulations surpasses the resources available from a normal single computer and either the use of a super-computer or distributing the simulation over a computer network has to be considered. The acquisition and/or use of a super-computer, however, is expensive, whereas distributing the simulation over a com-

puter network allows for the use of a range of relatively cheap computers, from resource rich desktop computers to the lowly PDA. Peer-to-peer networks allow for simulations of a scale relative to the number of nodes in the network but do not require the use of fast and/or bandwidth rich servers. In fact, peer-to-peer networks have been used to utilise unused resources on computers all over the Internet, much like in the SETI@home project (see Korpela et al. (2001) and Anderson et al. (2002) for more information). Although peer-to-peer networks are well-suited to handle large scale distributed simulations, they also pose some problems of the own.

For one, all distributed simulations must assume at least some level of unreliability in the availability of the resources in the network. Computer networks, however they are organised, consist of a collection of computers, and computers can become unavailable, can be removed from the network by the user, or can even fail. Although in modern computers and computer networks the failure-rate is small, if the number of computers in the network is large enough or the duration of the simulation long enough, some resource- or information-loss has to be expected. Several mechanisms have been proposed to limit the amount of uncertainty to which the simulation is exposed to, but all these have an adverse effect on the efficiency of the system. In general, we have to assume that a certain level of unreliability in the simulation is acceptable. This is summarised as the *unreliability assumption*.

3 Scalable grid landscapes and P2P networks

A grid landscape is a set of locations connected to each other so that together they form a grid pattern. A grid landscape is a convenient abstraction of the real-world and it is easy to implement and handle by a simulation run on a single computer. When a social agent simulation is distributed over a peer-to-peer computer network however, a grid landscape introduces a number of problems that are not apparent in a single computer implementation. Most, if not all, of these problems arise from the fact that the landscape has to be partitioned over the nodes of the peer-to-peer network. Partitioning is necessary since in a peer-to-peer network a server to handle this information centrally is not allowed: the landscape has to be handled collectively by the peer nodes of the network. In practise, partitioning the landscape means that each node in the peer-to-peer network is assigned a collection of locations that it will handle. Although it is convenient to think of these collections as clusters in the landscape, in practice, this might not be the most efficient assignment. In fact, in this paper we make no assumption about how the landscape is partitioned over the nodes of the network but instead we simply state that there exist several efficient methods for partitioning the landscape. These methods are often *self-organising* in order to maintain (at least an approximation of) the most efficient partitioning of the landscape (see Clarke et al. (2001) for more information). This means that they change the partitioning of the landscape dynamically during the run of the simulation in order, for example, to reflect changes in the network. This implies that the collection of locations handled by a single node in the network changes over time. This paper focusses on the problem of how to maintain scalability and consistency during the run of the simulation without adversely affecting the efficiency of the simulation too much.

3.1 Efficiency

The efficiency of a social agent simulation implemented on a peer-to-peer network depends to a large extent on the efficiency with which the underlying network can supply information that is needed. Environment information is requested frequently and requests — or queries — for landscape information have to be handled efficiency. In peer-to-peer terms, this means that the peer-to-peer network has to handle landscape information *discovery* efficiently. Query efficiency is measured by measuring the response-time, that is, the amount of time it takes between an issue of a query and the return of the requested information. In order to reduce query response-times, peer-to-peer networks use *information indexing*, a technique borrowed from database management systems (see Dabek et al. (2001) for more information on how information indexing is used with Chord). Indexing implies the generation and maintenance of redundant (meta) information, in order to more quickly locate pieces of information stored in the system. In peer-to-peer networks, indexing is used for two reasons: increasing network efficiency and increasing information discovery efficiency. An example of the first kind of indexing is maintaining a node-address index in order to speed-up direct communication between nodes. An example of the second kind of indexing is the maintenance of an index about certain kinds of information stored in the network in order to speed-up queries about this information. A balance has to be struck between the cost of maintaining the index and the speed-up it allows. For more in-

formation on how to improve data access in peer-to-peer systems and the use of indexing see Aberer et al. (2002).

3.2 Scalability

The scalability of the system indicates the capability of a system to increase performance under an increased load when resources are added. Scalability is a highly significant issue in databases, routers, networking, etc. A system whose performance improves proportionally to the amount of resources added to it, is said to be a scalable system. For a peer-to-peer network to be scalable the performance of the overall network has to increase proportionally to the combined resources of the nodes that are added to the network. Depending on the algorithm, peer-to-peer networks are considered to be inherently scalable, although they can be less so when the overhead imposed by the network, outweighs the resource addition to the network. In order to maintain a scalable peer-to-peer network, it is imperative that the overhead needed for maintaining the network is also scalable.

All computer networks, however they are structured, impose some amount of overhead to keep them functioning (efficiently). An example of the overhead imposed for maintaining a peer-to-peer network is the information needed to set up the direct communication between nodes (see Rowstron and Druschel (2003) for an example of how this can be done). Indexing the location of the nodes in the network is a common technique for maintaining this information efficiently (see previous section for more information about efficiency and indexing). As the number of nodes of a network increase however, there exists the possibility that the overhead of the system becomes so extensive, that to simply maintain it will take up all the newly added resources of an added node. Therefore, to allow for truly large peer-to-peer networks, the amount of overhead required to maintain them has to be minimised.

3.3 Consistency

For any simulation to provide reliable results, the consistency of the information needed to run the simulation is of paramount importance. The *unreliability assumption* already states that some inconsistency has to be tolerated when a simulation is distributed over a computer network. A peer-to-peer network itself, however, can create inconsistencies in the information stored on it. In order to explain this we have to look at how a peer-to-peer network stores information. We have already explained that a peer-to-peer network has to distribute or partition the information stored on it over its nodes. What we have not explained is how this is done. Information stored on a peer-to-peer network will normally propagate through the network until such time as the partitioning technique used determines where it will be stored (see Druschel and Rowstron (2001) and Jelasity et al. (2002) for more information). This propagation of information is done both as a load-balancing technique, i.e. making sure that the information "load" of each node is proportional to the resources available by that node, and as a way of determining if the newly available information is inconsistent with information already stored in the network. Load-balancing in a peer-to-peer network is done as part of the partitioning technique.

In a peer-to-peer distributed social agent simulation, in order to increase the scalability of the system, information about the landscape should be maintained as sparsely as possible. Only landscape locations that are needed should be maintained and the locations themselves should be created only when they are needed. In a peer-to-peer distributed social agent simulation, landscape information inconsistencies can occur whenever a new location of the landscape is created. This is because it is possible to create a location on one node that has already been created (or is created at the same time) on another. The underlying peer-to-peer network has to be able to handle these kinds of inconsistencies, for example, by keeping the earlier created location and dispensing with the newly created location. However, it has to first be aware that these inconsistencies exist, hence the need to propagate information about location creation through the network.

Propagation of information through a peer-to-peer network and the subsequent handling of any possible inconsistencies takes time. In some cases, the simulation itself is robust to some level of inconsistency of information, reducing the need handle the inconsistencies quickly. In other cases, inconsistent information can have serious consequences, placing more stringent requirements on the capabilities of the peer-to-peer network. In general however, the ability of the peer-to-peer network to handle inconsistencies incurs some overhead on the network and a balance between the efficiency and scalability of the system on the one hand and the ability to handle inconsistencies has to be found.

Just as with the *unreliability assumption*, an *inconsistency assumption* can be formulated, that is, a dis-

tributed system in which the information stored on it is totally consistent can only be created at great cost to the efficiency and/or scalability of the system. As such, any distributed system assumes that a certain level of inconsistency of information is inevitable and that the system has to deal with those inconsistencies accordingly. The unreliability and the inconsistency assumptions also have consequences for experiments that can be run on a distributed system, in that experiments in which completely consistent information is required cannot be run.

3.4 The move- and look-action

The delicate balance that has to be struck between scalability, consistency and efficiency in a peer-to-peer distributed social agent simulation involves two basic actions that an agent in a simulation can take: move and look. Most other actions, at least from an information requirement perspective, can be seen as variations on these two actions and they can be handled analogously to them.

The move-action allows an agent to move from one location in the landscape to another. In order to limit the amount of landscape information needed by the simulation, and thereby increasing the scalability of the system, the landscape should be maintained as sparsely as possible: only the locations needed are created and only the locations already created are stored. The move-action is important for the peer-to-peer network because when an agent moves to a new location — one that has not been created yet — the simulation has to create one for it. When two nodes in the peer-to-peer network need to create the same location at the same time, this can lead to inconsistency in the network. Handling the inconsistency at this time is complex and can include backtracking to an earlier state of the system or even the removal of an agent or other movable object from the simulation. We solve this problem through the use of a buffer-zone of locations around agents and other movable objects.

The look-action provides the agent with the perceptual information it needs to decide on the actions it is going to undertake. A look-action can be a conscious action for the agent to take but social agent simulations in which this information is given without a agent having to explicitly do a look-action exist as well. A look-action results in a number of information queries on the locations that the agent can see. Commonly, the number of locations visible to an agent is determined by a *look-distance* parameter and a *look-direction* of the agent. Together, they define a *look-arc* of locations visible to the agent. Since agents look often and sometimes can look far, the action produces a large amount of queries that all have to be handled efficiently. A query in a peer-to-peer network is handled by propagating it through the network. Each node with relevant information then participates towards resolving the query. In a peer-to-peer network without complete information, a query that can not be resolved takes a long time to fail, since it has to propagate throughout the whole network. In small peer-to-peer networks, the response-time, and thus the efficiency of the system, will be reasonable, but with the addition of more nodes, the system can become more and more inefficient. This straightforward implementation obviously is not scalable to a large peer-to-peer network. In this paper, we propose an alternative implementation involving predictive indexing of locations, so that the queries resulting from a look-action can be handled efficiently by the system without adversely affecting its scalability.

4 The move-buffer

The move-buffer should be seen as a buffer-zone around the locations in the landscape that are or have been used by the agents in the simulation. The buffer-zone is used as a means to ensure that consistency of information is maintained during the run of the simulation. Instead of creating locations at the moment when an agent wants to move to them, we create a buffer-zone of locations around all the locations that the agents or other movable objects have used before or are using now, so that in the case that the agents want to move into the buffer-zone, the locations are already created and the information about these locations is consistent throughout the network. The information about the locations in the buffer-zone can be propagated through the network so that the peer-to-peer network can handle any inconsistencies before an agent moves onto them.

An example of how an inconsistency can be avoided by using buffer-zones is when two agents on two landscape islands, handled by two nodes in the network, start moving toward each other. Each time one of the agents moves onto a location in the buffer-zone, the buffer-zone is extended in such a way that a certain, user-defined, distance around the agents is covered by the buffer-zone. At some point, the buffers of both nodes will include some of the same locations. At this point, an inconsistency can occur when two nodes try to create the same location at the same time. This inconsistency can be resolved by choosing one node to handle the location, and copying over information to the other node.

However, while the inconsistency is being resolved, the agents may keep moving. The buffer-zone has to cover enough locations around the agents so that the network has enough time to handle any inconsistencies that can occur before the locations in the buffer-zone are needed by the simulation.

Figure 3 shows an example of how the move-buffer is used. The small solid circles, like the one at coordinate $\langle D, 5\rangle$, indicate a location in the landscape that has been created and used by agents in the simulation. The small dashed circles, like the one at coordinate $\langle D, 6\rangle$, indicate a location in the move-buffer. The two X-s at coordinates $\langle E, 4\rangle$ and $\langle J, 4\rangle$ in the top diagram and $\langle E, 4\rangle$ and $\langle I, 4\rangle$ in the bottom diagram indicate two agents, agent X on the left and agent Y on the right. The simulation is partitioned over two nodes at the line in the middle of the two diagrams. The large thin circle indicates the buffer-zone around the agents. In the bottom diagram, agent Y has moved from coordinate $\langle J, 4\rangle$ to coordinate $\langle I, 4\rangle$. The buffer-zone is extended with the locations at $\langle I, 2\rangle$, $\langle I, 6\rangle$, $\langle H, 3\rangle$, $\langle H, 5\rangle$, and $\langle G, 4\rangle$. However, the location at coordinate $\langle G, 4\rangle$ is already in the buffer-zone of node 1, so there is a possibility that its creation has caused an inconsistency. This inconsistency has to be handled by the network, for example, by copying the information about location $\langle G, 4\rangle$ stored on node 1 over the information stored on node 2. Because the buffer-zone has a radius of two locations, the time allowed to the network for handling the information inconsistency is the same as the time it takes for the agents to make two moves towards each other.

The distance covered by the move-buffer is configurable for each experiment but should take two things into account: the size of the peer-to-peer network, and the level of inconsistency that the simulation can tolerate. The larger the network, the longer it takes for landscape information to propagate through it. When a inconsistency occurs, the time needed to propagate this message back also has to be taken into account. Some simulations are more robust to inconsistencies than others. In some, the landscape information does not have to be exact all the time. These simulations do not require an extensive buffer-zone. Other simulations however do require nearly exact knowledge about the environment, and then, the distance that their buffer-zone should cover, needs to be quite large. Since the move-buffer is part of the overhead of the system, the radius of the move-buffer has an effect on the scalability of the simulation. A larger buffer-zone means more information has to be propagated through, and stored on, the network and

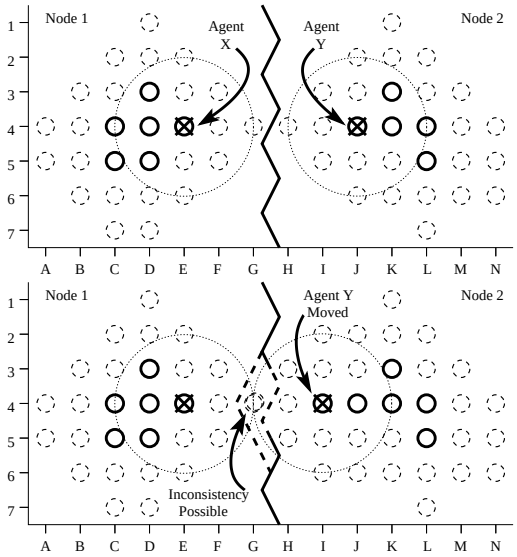

Figure 3: Move-buffer example

so more resources are needed to maintain this information. This becomes clear when we consider an extreme case, that is, when all possible locations in the landscape are in the buffer-zone at initialisation of the simulation. This will cause instantiation of the whole landscape, which depending on the maximum size of the landscape, could make the resource requirements of the network so large that it becomes impractical.

The distance that the buffer-zone covers therefore, should be configurable by the user of the simulation, because only the user can make an assessment of what kind of network is available, and what amount of inconsistency is acceptable. An interesting research question is how to assess and/or maintain the distance that the buffer-zone covers without user input.

5 The look-index

With the look-action, the efficiency of the system is the more important issue. When a peer-to-peer network has to handle a query it is resolved by propagating it through the network so that each node can provide information in order to resolve it. When the peer-to-peer network is large, it is possible that such a query will take a long time to resolve, thereby lowering the response-time of the system and thus its efficiency. When the query can not be resolved at all, for example when the necessary information to resolve it is not available in the network, the query has to propagate through the whole network and back, before its

failure can be reported. The worst-case response-time of a query for information in a peer-to-peer network is therefore proportional to the size of the network.

We assume that the queries resulting from a look-action occur often during the run of the simulation. As the landscape is stored as sparsely as possible, there will be (possibly large) portions of the landscape that have not been created yet. The agents in the simulation, however, might still want to look at these portions of the landscape, and since this landscape information is unavailable in the network, we must assume that a (possibly large) number of queries will consequently fail. As a result, the efficiency of a simulation will be adversely affected. In an effort to increase the efficiency of the peer-to-peer network, we propose to index information about the landscape. If a large enough portion of the landscape is indexed, all the information needed for a query should be available in the *look-index*.

In the look-index, we index the locations that the agents might want to look at. This implies that some level of prediction of where the agents might want to look is possible. The index is generated by adding either an empty entry to the index for locations that have not yet been created or used, or the address of the node where the location is handled if it has been created or used. Instead of propagating information about the location through the network, we propagate only the newly created index entries through the network. The node in the network that handles the location then sends back its address to the node that sent the index entry. When an agent does a look-action, before any queries are sent through the network, the simulation first inspects the index. If the location corresponds to an empty index entry in the index, the network knows that the location has not been created yet and no query is issued for that location. If the location corresponds to a non-empty index entry, the network can use the node-address to set-up a direct network connection to retrieve the information about the location efficiently.

The look-index is maintained so that the information that a query can request is complete, in order to make sure that no query can fail. The size of the look-index is a trade-off between the efficiency of the simulation and its scalability. This is best explained by looking at the two extremes of the extent of the look-index. On the one hand, the look-index can extend over the whole possible landscape. This will provide complete information about the whole landscape but also means that the whole landscape needs to be indexed on every node of the simulation. Maintaining an index of the whole landscape on every node is clearly not a scalable solution, especially when large landscapes are used. The other extreme is to maintain no index at all. Although this certainly is a scalable solution, the efficiency of the simulation will suffer as a large portion of the queries issued to the network will probably fail and will take a long time do so.

The ideal extent of the index lies somewhere in between these two extremes and must be set by the user of the simulation, as only the user is able to estimate how much incomplete information is allowable and how scalable the network has to be. As a rule of thumb, however, we suggest that the look-index extends a few locations farther that the distance that the agents can look. The extra extent beyond the look-distance allows the peer-to-peer network some time to propagate the index entries through the network and also so that when two entries about the same location are created in separate parts of the network, the network can validate that the two index entries do not contain conflicting information.

6 Conclusions

In this paper we identified that when a grid landscape is used as a landscape representation in a peer-to-peer distributed social agent simulation, the scalability, information consistency, and efficiency of the simulation can be adversely affected when a naive implementation is used. Based on two basic agent actions — move and look — we demonstrate that a trade-off between these properties is possible when grid locations in the landscape are included in a buffer-zone, in the case of the move-action, or indexed, in the case of the look-action. The extents of the buffer-zone and the index should be set by the user of the simulation, since only the user can have full knowledge of the size of the network and the amount of inconsistency that is acceptable. Future research includes developing methods to estimate both parameters, either by providing an estimation at initialisation or by adjusting the parameters during the simulation itself.

Acknowledgements

The NewTies project is partially supported by the European Commission Framework 6 Future and Emerging Technologies programme under contract 003752. Opinions are the authors' own and do not represent those of the Commission. The ideas in this paper have greatly benefited from interaction with the partners in the project (Vereniging voor Christelijk Wetenschappelijk Onderwijs; Eötvös Loránd University;

Napier University; University of Surrey and Stichting Katholieke Universiteit Brabant).

References

Karl Aberer, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. Improving data access in p2p systems. *IEEE Internet Computing*, 6(1), Jan/Feb 2002.

D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D Werthimer. Seti@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

I. Clarke, O. Sandbarg, B. Wiley, and T.W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In H. Federrath, editor, *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *Lecture Notes on Computer Science*. Springer: New York, 2001.

F. Dabek, E. Brunskill, F. Kaashoek, D. Karger, R. Morris, I. Stoica, and H. Balakrishnan. Building peer-to-peer systems with chord, a distributed lookup service. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HOTOS-VIII)*, 2001.

Peter Druschel and Antony Rowstron. Past: Persistent and anonymous storage in a peer-to-peer networking environment. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS VIII)*, pages 65–70. Schloss Elmau, Germany, May 2001.

M. Jelasity, M. Preuß, and B Paechter. Maintaining connectivity in a scalable and robust distributed environment. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid*, page 389, 2002.

E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M Lebofsky. Seti@home — massively distributed computing for seti. *IEEE Computational Science and Engineering*, 3(1):78–83, Jan/Feb 2001.

Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218(2001):329, Jun 2003.