Integrating Peer-to-Peer Networking and Computing in the AgentScape Framework

Benno J. Overeinder, Etienne Posthumus, and Frances M.T. Brazier
Department of Computer Science, Faculty of Sciences, Vrije Universiteit Amsterdam
de Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
{bjo,etienne,frances}@cs.vu.nl

Abstract

The combination of peer-to-peer networking and agent-based computing seems to be a perfect match. Agents are cooperative and communication oriented, while peer-to-peer networks typically support distributed systems in which all nodes have equal roles and responsibilities. AgentScape is a framework designed to support large-scale multi-agent systems. Pole extends this framework with peer-to-peer computing. This combination facilitates the development and deployment of new agent-based peer-to-peer applications and services.

1 Introduction

The emergence of fast wide-area networks has moved distributed computing to a new area in which distributed resources in a wide-area network are more closely coupled to provide a single integrated platform for computing and data storage. Well-known examples are Grid computing and peer-to-peer networks, which harness the computing power of hosts in a network and make their under-utilized resources available to users.

Informally, peer-to-peer systems can be described to be distributed systems in which all nodes are peers in the sense that they have equal roles and responsibilities. The nodes in the distributed system have identical capabilities and responsibilities, and all communication is symmetric. Peer-to-peer systems are characterized by decentralized control, large scale, and extreme dynamics of their operating environment.

Peer-to-peer systems are used for file-sharing (i.e., exchanging files between peers) and file-storage (i.e., the peer-to-peer network is used as a distributed file system) [4, 25]. But peer-to-peer networks can also be used to harness the computing power of hosts in a network, similar to SETI@home or more elaborate systems like Condor [19] and Grid computing environments [9].

In multi-agent systems, the (basic) interaction pattern between agents is peer-to-peer. Autonomous agents can observe their environment and reason and act on the basis of these observations [35]. Agents are often adaptive: adaptive to new environments, adaptive to new structures.

The main goal of integrating peer-to-peer functionality within the AgentScape framework for large-scale multi-agent systems is to simplify peer-to-peer application and service development and deployment by freeing the programmer of all low-level details including communication, security, and scheduling.

This paper briefly describes the AgentScape framework and the Pole system: an extension to the AgentScape framework for basic peer-to-peer networking and computing support. The peer-to-peer support is incorporated at three levels: the middleware layer, the service layer, and at the application level (the API runtime libraries). Section 2 briefly introduces some concepts in peer-to-peer computing and autonomous agents. The AgentScape framework is presented in Section 3, and the integration of peer-to-peer networking and computing support is explained in Section 4. Section 5 concludes the paper with a discussion and future work.

2 Peer-to-Peer Networking and Autonomous Agents

This section presents a concise overview of peer-to-peer networking and autonomous agents.

2.1 Peer-to-Peer Networking

Peer-to-peer computing has offered a compelling and intuitive way for users to find and share resources directly with each other, often without requiring a central authority or server. Although today's applications are primarily used for finding, retrieving, and using information, they hint at what complete access to the Internet can deliver in

the future. Resources, including information and processing power, can be shared directly from those who have them to those who need them. Buyers and sellers can be matched directly through P2P auction and transaction services, and new approaches to distributed resource-sharing like SETI@home, Condor [19], or Grid environments [9] are appearing. Peer-to-peer computing enables applications that are collaborative and communication-focussed: it leverages available computing performance, storage, and bandwidth found on systems connected to each other in a world-wide network.

Today's most well-known peer-to-peer applications are Napster, Gnutella [4], and Freenet [5], but various research projects have been initiated in the past few years, such as Pastry [27] and Chord [28]. Although the different peer-to-peer applications share the same notion of peer-to-peer networking, the intended usage and approach varies from application to application.

Napster and Gnutella are primarily file-sharing applications: exchange of files between peers. Napster's approach to information search is traditionally client-server, while Gnutella adheres more to the peer-to-peer philosophy and forwards information search requests to its neighboring peers in the network. (Although Gnutella recently introduced super nodes and client nodes for more scalable information retrieval.) Freenet is more like a distributed information storage system. It pools unused disk space across potentially hundreds of thousands of desktop computers to create a collaborative virtual file system.

Pastry provides a scalable, distributed object location and routing infrastructure for wide-area peer-to-peer applications. It can be used to support a variety of peer-to-peer applications, including global data storage, data sharing, group communication and naming. Chord, on the other hand, focuses on a scalable peer-to-peer lookup service to efficiently locate the node that stores a particular data item. Chord provides support for just one operation: given a key, it maps the key onto a node.

The JXTA project from Sun Microsystems [10, 29] works on core network computing technology to provide a set of simple, small, and flexible mechanisms that can support peer-to-peer computing. The focus is on creating basic mechanisms and leaving policy choices to application developers.

The self-organizing behaviour of peer-to-peer networks has also been studied. In particular scalability, fault-tolerance, and security have been subject of study. It has been observed that peer-to-peer networks organize themselves into a "small-world" networks [5, 16], which are typically characterized by a power-law distribution of the edge degree. In such a distribution, the majority of nodes have relatively few local connections to other nodes, but a significant small number of nodes have large wide-ranging sets of connections. Even in very large networks, the small-world topology enables short paths because these well-connected nodes provide shortcuts [33]. Small-world networks are surprisingly resistant to random errors, because random failures are most likely to eliminate nodes from the poorly connected majority of nodes. But the feature that makes it immune to accidents also makes it vulnerable to attacks if the well-connected nodes are targeted.

2.2 Autonomous Agents

From a Computer Systems perspective an agent is a process, a piece of running code with data and state. The functionality of these agents can most often be described in terms of human behaviour, and to which the predicate intelligent is associated [35]. Agents are processes that are autonomous and pro-active (capable of making "their own" decisions when they like), that can interact with objects and services, communicate with other agents and may be mobile.

Agents interact with objects. Objects are passive [15]. In other words, an object needs to be invoked in order to perform a function, and performs only during an invocation. Agents, on the other hand, receive messages and autonomously decide if, when, and how to (re-)act. The only way for one agent to influence another agent is by sending a message, possibly with a request. An agent is free to ignore or react to such requests.

Message delivery may be subject to different "quality of service" levels. For example, the message paradigm described by FIPA prescribes reliable and ordered point-to-point communication between agents [8].

Agents in computer systems are often mobile. The decision to migrate is taken autonomously by a mobile agent itself. The ability of migration provides mobile agents a means to overcome the high latency or limited bandwidth problem of traditional client-server interactions by moving their computations to required resources or services. The current evolution of intelligent and active networks in system and network management, for example, is based on this technology [3]. A similar tendency is observed in the search and filtering of globally available information such as in the electronic marketplaces, e-commerce, and information retrieval on the World Wide Web [13].

A distinction can be drawn based on whether the execution state is migrated along with the unit of computation or not [24]. Systems providing the former option are said to support *strong mobility*, as opposed to systems that discard the execution state across migration, and are hence said to provide *weak mobility*. In systems supporting strong mobility, migration is completely transparent to the migrated program, whereas with weak mobility, extra programming is required in order to save part of the execution state.

Strong mobility as found in NOMADS [30], Ara [23], and D'Agents [12], requires that the entire state of the agent, including its execution stack and program counter [14], is saved before the agent is migrated to its new location. Strong mobility is a complicated task to realize, and typical implementations of this functionality in multi-agent platforms as mentioned above, provide platform specific solutions. As a consequence, interoperability between heterogeneous multi-agent systems is difficult, if not impossible, to realize.

Many of the multi-agent platforms support weak mobility (like Ajanta [31] and Aglets [18]). Most of the agent systems are implemented on top of the Java Virtual Machine (JVM), which provides with object serialization basic mechanisms to implement weak mobility. The JVM does not provide mechanisms to deal with the execution state.

Security is of great importance in agent systems, not only in electronic monetary transactions, but also that mobile agents should not become the next generation of viruses. Current research on secure agent systems concentrates mainly on protecting hosts against hostile mobile agents. The problem of security stems from the fact that untrusted code needs to be executed. Modern solutions are based on the notion of protection domains by which a security policy for accessing local resources can be enforced [11]. Only very few systems also provide facilities for protecting mobile agents against hostile hosts [17, 26].

3 The AgentScape Framework: A Scalable Multi-Agent Infrastructure

AgentScape is a middleware layer that supports large-scale agent systems. The rationale behind the design decisions are (i) to provide a platform for large-scale agent systems, (ii) support multiple code bases and operating systems, and (iii) interoperability with other agent platforms [34]. The consequences of the design rationale with respect to agents and objects, interaction, mobility, security and authorization, and services are presented in the following subsections. This section concludes with some notes on a prototype implementation of the AgentScape model.

3.1 The AgentScape Model

The overall design philosophy is "less is more," that is, the AgentScape middleware should provide a minimal but sufficient support for agent applications, and "one size does not fit all," that is, the middleware should be adaptive or reconfigurable such that it can be tailored to a specific application (class) or operating system/hardware platform.

Agents and *objects* are basic entities in AgentScape. A *location* is a "place" at which agents and objects can reside (see Fig. 1). Agents are active entities in AgentScape that

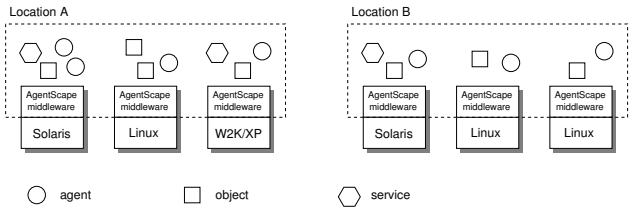

Figure 1. AgentScape conceptual model.

interact with each other by message-passing communication. Furthermore, agent migration in the form of weak mobility is supported. Objects are passive entities that are only engaged into computations reactively on an agent's initiative. Besides agents, objects, and locations, the AgentScape model also defines *services*. Services provide information or activities on behalf of agents or the AgentScape middleware.

Scalability, heterogeneity, and interoperability are important principles underlying the design of AgentScape. The design of AgentScape includes the design of agents, objects and services, interactions, migrations, security and authorization, as well as the agent platform itself. For example, scalability of agents and objects is realized by distributing objects according to a per-object distribution strategy, but not agents. Instead, agents have a public representation that may be distributed if necessary.

The basic idea in the AgentScape model is that the core functionality is provided by the *agent interface* implementations such that the middleware (or the agent representation of the middleware) can be designed to perform basic functions. This approach has a number of advantages. As the middleware provides basic functionality, the complexity of the design of the middleware can be kept manageable and qualities like robustness and security of the middleware can be more easily asserted. Additional functionality can be implemented in the agent-specific interface implementation (see Fig. 2).

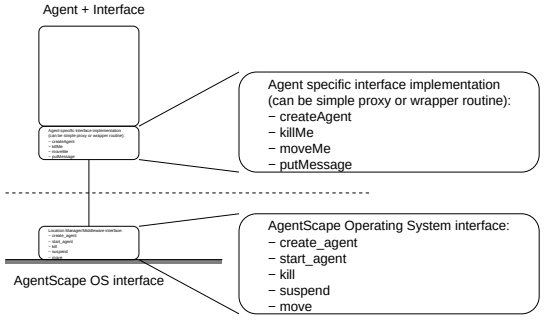

Figure 2. The AgentScape interface model.

Agent-agent interaction is exclusively via message-passing communication. Asynchronous message passing has good scalability characteristics with a minimum of synchronization between the agents.

Agent migration between locations is based on weak mobility [24] (see also Section 2.2). The *state* of the agent is captured (e.g., the variables referenced by the agent) but not the *context* of the agent (e.g., stack pointer and program counter).

3.2 An AgentScape Architecture

Agents and objects are supported by *agent servers* and *object servers* respectively. Agent servers provide the interface and access to AgentScape to the agents that are hosted by the agent server. Similarly, objects servers provide access to the objects that are hosted by the object server. Services are made accessible in AgentScape by service access providers.

A location is a closely related collection of agent and object servers, possibly on the same (high-speed) network, on hosts which are managed in the same administrative domain. Each host runs a *minimal* AOS kernel, and zero or more agent servers, objects servers, and service access providers. A location is implemented by the distributed AOS kernels, the agent servers, the object servers, and service access providers.

Depending on the policy or resource requirements, one agent can be exclusively assigned to one agent server, or a pool of agents can be hosted by one agent server. The explicit use of agent servers makes some aspects in the life cycle model of agents more clear. An active agent is assigned to, and runs on a server; a suspended agent is not assigned to an agent server. In this model, starting a newly created, or activating an existing suspended agent, is similar, and some design decisions of the agent life cycle can be simplified.

The use of agent and object servers is transparent to the agents. Hence from the agent perspective, agent servers do not belong to the AgentScape model. However, e.g., for performance or security management reasons, an agent could ask the middleware to determine on which agent server the agent runs.

The *AgentScape Operating System* (AOS) forms the basic fundament of the AgentScape middleware. An overview of the AgentScape architecture is shown in Fig. 3. The AOS offers a uniform and transparent interface to the underlying resources and hides various aspects of network environments, communication, operating system, access to resources, etc. The AgentScape API is the interface to the middleware. Both agents and services (e.g., resource management and directory services) use this interface to the AOS middleware.

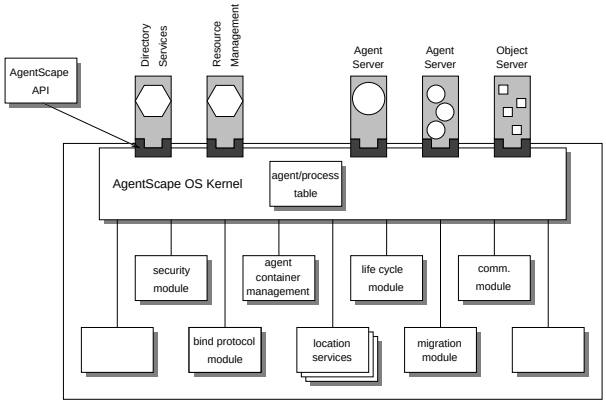

Figure 3. An AgentScape middleware architecture.

The design of the AgentScape Operating System is *modular*. The AOS kernel is the central active entity that coordinates all activities in the middleware. The modules in the AOS middleware provide the basic functionality. Below a brief overview of the most important modules is given. The life-cycle module is responsible for the creation and deletion of agents. The communication module implements a number of communication services, e.g., similar to UDP, TCP, and streaming, with different qualities-of-service. Support for agent mobility is implemented in the migration module. The location service associates an agent identifier with an contact-address. There are also location services for objects, services, and locations. The security architecture is essential in the design of AgentScape, as it is an integral part of the middleware. Many modules in the middleware have to request authentication or authorization in order to execute their tasks.

In AgentScape, interoperability between agent platforms can be realized in two ways. First by conforming to standards like FIPA [8] or OMG MASIF [20]. These agent platform standards define interfaces and protocols for interoperability between different agent platform implementations. For example, the OMG MASIF standard defines agent management, agent tracking (naming services), and agent transport amongst others. The FIPA standard is more comprehensive in that it defines also agent communication and agent message transport, and even defines an abstract architecture of the agent platform. A second approach to interoperability is realized by reconfiguration or adaptation of the mobile agent. This can be accomplished by an agent factory as described by Brazier et al. [1], which regenerates an agent given a blueprint of the agent's functionality and its state, using the appropriate components for interoperability with the other agent platform.

3.3 AgentScape Prototype

A prototype implementation of the AgentScape architecture is currently available and provides the following basic functionality: creation and deletion of agents, communication between agents and middleware, and weak migration of agents. The AgentScape Operating System kernel and some basic services are implemented in the programming language Python, while the agent servers are implemented in Java. Agent servers for other programming languages will be made available in forthcoming releases of AgentScape.

Distributed shared (replicated) objects in AgentScape will be supported by the Globe system [32]. Globe is a large-scale wide-area distributed system that provides a object-based framework for developing applications.

The use of multiple programming languages is not only available at the application level (i.e., building agents and objects in a programming language of choice), but also the modules of the AOS are implemented in different programming languages. For example, multiple location services can be present in the AOS, each implemented in a different language. One specific implementation of a location service based on peer-to-peer networking is described in the next section.

4 Integrating Peer-to-Peer Networking and Computing in AgentScape

This section presents the proposed software architecture of the AgentScape peer-to-peer networking and computing infrastructure. The peer-to-peer networking infrastructure in AgentScape is called Pole*. The design approach of Pole lies somewhere in between Chord and Pastry, and JXTA (as described in Section 2.1). Similar to Chord and Pastry, Pole implements an overlay network for routing messages and storing information. However, the intended design of Pole also includes high-level services for more complex coordination using peer-to-peer networks, and making these services available to the peer-to-peer application developer; similar to the JXTA project philosophy.

The peer-to-peer support in AgentScape is intended for the development of a variety of agent-based peer-to-peer Internet systems like global file sharing, file storage, name and location services, group communication, and agent coordination. Typical applications that can use the AgentScape peer-to-peer infrastructure are distributed auctions and information retrieval. These applications put stringent requirements on agent coordination and can profit from the peer-to-peer functionality provided. Other appliance of peer-to-peer functionality can be in instant messaging or, on a system level, in agent-based resource management on wide-area networks [22].

*Peer is consonant with pier, which is build on top of poles.

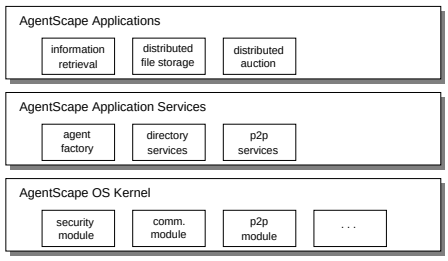

Figure 4. AgentScape peer-to-peer architecture.

The Pole peer-to-peer software architecture can be roughly broken down into the three layers as presented in the AgentScape framework: AOS kernel layer, services layer, and application layer (see Fig. 4). The AOS kernel layer deals with communication management such as routing and other low-level "plumbing." The service layer deals with higher-level concepts, such as indexing, searching, and file sharing. At the top is the layer of applications such as instant messaging, information retrieval, auctioning, and storage systems.

4.1 AgentScape OS Kernel Support for Peer-to-Peer Networking

The AgentScape OS kernel provides basic support for services and agent applications. The AOS kernel can be extended with peer-to-peer functionality by loading (either at startup or dynamically) the p2p module into the AgentScape middleware (see Fig. 4). Modules in the AgentScape middleware implement the AgentScape OS kernel functionality. The functionality provided by the modules is available to the layers built on top of the AOS kernel layer, e.g., services and applications make use of the security module for implementing their security policy or encrypting their messages that are sent to and fro.

At the AOS kernel layer, mechanisms for peer groups are supported to create policies for creation and deletion, communication, and security. The current prototype implementation of AgentScape uses XML-RPC to accomplish message-passing communication. The messages sent between peers are structured with XML, and support transfer of data, content and code in a protocol-independent manner. Future developments also require monitoring of the behaviour and activity of peers, in order to implement access control, priority setting, traffic metering, and bandwidth balancing.

The current p2p kernel module provides the basic mechanisms to realize a peer-to-peer overlay network and message routing. The first Pole prototype implementation of the

p2p module organizes the hosts in an overlay network with a circle topology, i.e., a host is a node on the circle topology. Currently, the MD5 digest (128-bit hash) of the IP number of a host is computed and the numerical value of the MD5 IP hash is used determine its node position on the circle topology. The MD5 digest of a combination of an IP number and a port number can be used to introduce virtual nodes, where multiple virtual nodes can reside on one single host to improve load balancing (e.g., servers with large amounts of memory and high bandwidth connectivity can host multiple virtual nodes, while a laptop connected by WLAN hosts one node).

The index keys are stored in the peer-to-peer network at the node for which the direct successor relation holds. For example, suppose the numerical value of the index key is k, then the index key and the associated data is stored at node a (the MD5 hash of the IP number), where $a \leq k$, and there is no other node b such that $a < b$ and $b \leq k$. An important characteristic of hash functions, and in particular for secure hash functions like SHA-1, is that the hash function maps index keys (or the IPs and port numbers in case of hosts) to hash values with a uniform distribution. This characteristic also ensures a uniform distribution of the index keys over the nodes in the network.

The search algorithm for keys in the peer-to-peer network makes use hypercube routing (or generalized prefix routing) over the circle topology. The 128-bit hash id's of the nodes are used for prefix routing: given the key index value, the search request is sent to the node that halves the distance to the destination node. To this end, each node keeps a table of node id's at $2^0, 2^1, \ldots, 2^{m-1}$ distance in the m-bit (in our case 128-bit) address space.

The self-organizing peer-to-peer overlay networks based on distributed hash tables provide efficient and fault-tolerant routing, object location, and load balancing. However, the basic protocols do not consider network proximity at all. An extension to the basic Pastry routing protocol is a simple heuristic that measures the proximity among a small number of nodes. The resulting local properties are used for routing messages to reduce network usage [2]. Chord reduces lookup latencies by preferential contacting nodes likely to be nearby in the underlying network. To this end, Chord estimates the routing costs to neighboring nodes based on latencies observed while building its routing tables [7]. Similar approaches will be adapted by Pole.

4.2 AgentScape Services for Peer-to-Peer Computing

AgentScape services for peer-to-peer computing expand upon the capabilities of the AOS kernel and facilitate application development. Facilities provided in this layer include mechanisms for searching, sharing, indexing, and caching. Search capabilities can include distributed, parallel searches across peer groups. Note that the directory services in AgentScape can make use of the peer-to-peer services. Directory services are typically databases (or data structures) that can be indexed by attributes, and their distributed information can be updated using peer-to-peer mechanisms.

One specific service currently under development in the AgentScape environment is a location service based on peer-to-peer networking. A location service in AgentScape maps agent handles to contact-addresses. An agent handle in AgentScape is a location service specific data structure used to efficiently map a unique agent identifier to its contact-address. AgentScape can support multiple location services, for example one based on DNS (or something similar with a hierarchical structure) and another based on peer-to-peer networking. The unique agent identifier is registered at a specific location service, and a location service specific agent handler is returned for future use.

The peer-to-peer location service prototype computes the MD5 digest value of the agent's identifier and uses this value as the agent handle (the key used for searching in the peer-to-peer network). The hash value of the agent identifier and the contact-address are stored in the peer-to-peer network as described in Section 4.1. The contact-address can be compared with a business card stating different protocols and references to contact a person, e.g., address followed by street, number, and city; telephone followed by a number; email followed by an email address; or TCP/IP followed by IP and port number. When an agent wants to contact another agent, it gives the agent handle to the location service, and the contact-address is returned. Given the contact-address, the agent can now choose between the different protocols to contact the remote agent.

In general, the peer services layer can be used to support other custom, application-specific functions, for example a secure peer messaging system could be built to allow anonymous authorship and a persistent message store. The peer services layer provides the mechanisms to create such secure tools; specific tool policies are determined by the application developers themselves.

4.3 Peer-to-Peer AgentScape Applications

AgentScape agent-based peer-to-peer applications are built using the peer services as well as the AOS kernel layer. The Pole peer-to-peer facilities are made available to the application developer as runtime libraries that implement the API to AgentScape peer-to-peer services and the AOS kernel peer-to-peer module.

Peer applications enabled by both the core and peer services layers include peer-to-peer electronic auctions and markets that link buyers and sellers directly [21]. Agent-based resource sharing applications can be built more

quickly and easily.

Besides the AgentScape interface to Pole, an interface to CP2PC will be provided. CP2PC is a minimal programming interface to file sharing peer-to-peer systems [6]. Client side applications can be built on top of this interface by different developers. CP2PC defines interfaces for joining, searching, downloading, publishing operations, and for managing meta-information, configuration and control, monitoring, and peer groups. The client side applications can be interfaced by CP2PC to other peer-to-peer systems like Pole, Chord, Pastry, JXTA, etc.

5 Summary and Future Work

This paper presents the Pole software architecture to extend the AgentScape framework with peer-to-peer computing facilities. The peer-to-peer computing functionality in the AgentScape framework is incorporated at three levels: AOS kernel, AgentScape services, and API runtime libraries. These peer-to-peer facilities can be used as building blocks in the construction of a variety of agent-based peer-to-peer Internet applications like distributed auctions, global file storage, and distributed information retrieval.

There is a prototype implementation of the Pole peer-to-peer software architecture and the AgentScape environment. One implementation of the location service in AgentScape makes use of the Pole peer-to-peer system. Other services in the AgentScape OS, such as name and directory services, will also make use of the peer-to-peer technology. Furthermore, the proposed management system in AgentScape used for allocating resources and load-sharing and balancing of agents over the resources, is based on agent-oriented peer-to-peer computing. The dynamic nature of available and unavailable resources, and dynamically created, deleted, and migrating agents, requires scalable and robust mechanisms to manage the distributed system effectively. Peer-to-peer computing seems to have the potential to offer a solution.

Future work on the Pole system will include key and data replication strategies for fault-tolerance and search performance. Other improvements will include the exploitation of network proximity in message routing in order to reduce communication latencies and increase bandwidth.

Acknowledgements

The authors would like to thank Niek Wijngaards and Maarten van Steen for the valuable discussions and their comments. This work is supported by NLnet Foundation, `http://www.nlnet.nl/`.

References

[1] F. M. T. Brazier, B. J. Overeinder, M. van Steen, and N. J. E. Wijngaards. Agent factory: Generative migration of mobile agents in heterogeneous environments. In *Proceedings of the ACM Symposium on Applied Computing (SAC 2002)*, pages 101–106, Madrid, Spain, Mar. 2002.

[2] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Exploiting network proximity in distributed hash tables. In *Proceedings of the International Workshop on Future Directions in Distributed Computing (FuDiCo 2002)*, Forlì, Italy, June 2002.

[3] W.-S. E. Chen and C.-L. Hu. A mobile agent-based active network architecture for intelligent network control. *Information Sciences*, 141(1-2):3–35, Mar. 2002.

[4] D. Clark. Face-to-face with peer-to-peer networking. *Computer*, 34(1):18–21, Jan. 2001.

[5] I. Clarke, S. G. Miller, T. W. Hong, O. Sandberg, and B. Wiley. Protecting free expression online with Freenet. *IEEE Internet Computing*, 6(1):40–49, Jan./Feb. 2002.

[6] CP2PC Home Page. http://www.cs.vu.nl/pubs/globe/cp2pc.

[7] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 202–215, Banff, Canada, Oct. 2001.

[8] J. Dale and E. Mamdani. Open standards for interoperating agent-based systems. *Software Focus*, 2(1):1–8, Spring 2001.

[9] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Fransisco, CA, 1999.

[10] L. Gong. JXTA: A network programming environment. *IEEE Internet Computing*, 5(3):88–95, May/June 2001.

[11] L. Gong and R. Schemers. Implementing protection domains in the Java Development Kit 1.2. In *Proceedings of the Symposium on Network and Distributed System Security*, pages 125–134, San Diego, CA, Mar. 1998. Internet Society.

[12] R. S. Gray, G. Cybenko, D. Kotz, R. A. Peterson, and D. Rus. D'Agents: Applications and performance of a mobile-agent system. *Software: Practice and Experience*, 32(6):543–573, May 2002.

[13] V. N. Gudivada, V. V. Raghavan, W. I. Grosky, and R. Kasanagottu. Information retrieval on the World Wide Web. *IEEE Internet Computing*, 1(5):58–68, Sept./Oct. 1997.

[14] K. A. Iskra, F. van der Linden, Z. W. Hendrikse, B. J. Overeinder, G. D. van Albada, and P. M. A. Sloot. The implementation of Dynamite: An environment for migrating PVM tasks. *Operating Systems Review*, 34(3):40–55, July 2000.

[15] N. R. Jennings and W. J. Wooldridge, editors. *Agent Technology: Foundations, Application, and Markets*. Springer-Verlag, Berlin, Germany, 1998.

[16] M. A. Jovanovic. Modeling peer-to-peer network topologies through "small-world" models and power laws. In *Proceedings of the IX Telecommunications Forum (TELFOR 2001)*, Belgrade, Yugoslavia, Nov. 2001.

[17] N. Karnik and A. Tripathi. Security in the Ajanta mobile agent system. *Software: Practice and Experience*, 31(4):301–329, Apr. 2001.

[18] D. B. Lange, M. Oshima, G. Karjoth, and K. Kosaka. Aglets: Programming mobile agents in Java. In *Worldwide Computing and Its Applications*, volume 1274 of *Lecture Notes in Computer Science*, pages 253–266. Springer-Verlag, Berlin, Germany, 1997.

[19] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor—A hunter for idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 104–111, San Jose, CA, June 1988.

[20] D. Milojicic et al. MASIF: The OMG mobile agent system interoperability facility. In *Proceedings of the 2nd International Workshop on Mobile Agents*, volume 1477 of *Lecture Notes in Computer Science*, pages 50–67, Berlin, Germany, Sept. 1998. Springer-Verlag.

[21] E. Ogston and S. Vassiliadis. A peer-to-peer agent auction. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002)*, Bologna, Italy, July 2002.

[22] B. J. Overeinder, N. J. E. Wijngaards, M. van Steen, and F. M. T. Brazier. Multi-agent support for Internet-scale Grid management. In *Proceedings of the AISB'02 Symposium on AI and Grid Computing*, pages 18–22, London, UK, Apr. 2002.

[23] H. Peine. Application and programming experience with the Ara mobile agent system. *Software: Practice and Experience*, 32(6):515–541, May 2002.

[24] G. P. Picco. Mobile agents: An introduction. *Microprocessors and Microsystems*, 25(2):65–74, Apr. 2001.

[25] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. Maintenance-free global data storage. *IEEE Internet Computing*, 5(5):40–49, Sept./Oct. 2001.

[26] V. Roth and M. Jalali-Sohi. Concepts and architecture of a security-centric mobile agent server. In *Fifth International Symposium on Autonomous Decentralized Systems (ISADS 2001)*, pages 435–443, Dallas, TX, Mar. 2001.

[27] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350, Berlin, Germany, 2001. Springer-Verlag.

[28] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'01)*, pages 149–160, San Diego, CA, Aug. 2001.

[29] Sun Microsystems, Inc. Project JXTA: An open, innovative collaboration. White Paper, http://www.jxta.org/project/-www/docs/OpenInnovative.pdf, Apr. 2001.

[30] N. Suri, J. M. Bradshaw, M. R. Breedy, P. T. Groth, G. A. Hill, R. Jeffers, T. S. Mitrovich, B. R. Pouliot, and D. S. Smith. Nomads: Toward a strong and safe mobile agent system. In *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 163–164, 2000.

[31] A. Tripathi, N. Karnik, M. Vora, T. Ahmed, and R. Singh. Mobile agent programming in Ajanta. In *Proceedings of the 19th International Conference on Distributed Computing Systems (ICDCS'99)*, pages 190–197, Austin, TX, May 1999.

[32] M. van Steen, P. Homburg, and A. S. Tanenbaum. Globe: A wide-area distributed system. *IEEE Concurrency*, 7(1):70–78, Jan.–Mar. 1999.

[33] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998.

[34] N. J. E. Wijngaards, B. J. Overeinder, M. van Steen, and F. M. T. Brazier. Supporting Internet-scale multi-agent systems. *Data and Knowledge Engineering*, 41(2–3):229–245, June 2002.

[35] M. J. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, June 1995.