Heterogeneous Mobile Agents for Distributed Processing

R Ghanea-Hercock, J C Collis, D T Ndumu

Intelligent Systems Research Group
BT Laboratories, Martlesham Heath, Ipswich, IP5 3RE, UK.
{ghanear,jaron,ndumudt}@info.bt.co.uk

Abstract

This paper presents a possible architecture for applying mobile agents to distributed high performance tasks. To achieve the required degree of scalability and automation future networks will require, careful consideration of the agent architecture is necessary. Our approach to mobile agent distributed processing is a two-tier management system. The primary goals are to hide the process of task distribution and to automate the management of the distributed system. At the global level, an anchored centralised agent is responsible for managing user interaction and determining how tasks are to be distributed. Whilst a group of task specific mobile agents deliver code to remote machines. We therefore developed specialised mobile agents, each performing particular roles, which co-operate as a *team* to achieve user defined goals. The resulting system provides users with a simpler means of utilising the power of distributed parallel computing.

1. Introduction

The potential power of networked computation resources has become apparent with the emergence of very large Intranets with high bandwidth links. PVM and similar systems [7] attempt to harness this potential, providing a means of combining distributed computing resources into a single virtual computer, usually termed a distributed virtual machine (DVM).

In its simplest form, distributed parallel processing (DPP) involves a 'manager' process that launches processes on different machines around a network, (in PVM control is distributed but a Master Daemon maintains the current configuration, [7]).

This paper presents current work on a mobile agent system, which is designed to provide a general-purpose architecture for automated management and distribution of computationally intensive code. The key element is the use of a heterogeneous architecture, i.e. several types of agent act co-operatively, and with varying degrees of mobility.

1.1 The Interaction Problem

Probably the biggest reason why DPP is not more widely used is the difficulty in configuring and managing distributed computational resources. Ideally the complexities of how, where and why the processes are running should be hidden behind a high-level interface. The desire for transparent distributed processing has motivated Microsoft's Millennium project [13], which is attempting to create an efficient, user-friendly, virtual machine system:

> "*Millennium raises the level of abstraction to create a self-tuning, self-configuring system in which programmers are not even aware of distribution; it happens entirely automatically.*" [13]

A similar objective has motivated Sun MicroSystem's Jini JavaSpace network computing framework, [22]. Jini anticipates a changing distributed computing environment with new services, hardware and protocols emerging; ideally DVMs should be able to accommodate and adapt to these developments.

Section 2, considers how mobile agent technology might improve the efficiency of DPP systems. Our approach to parallel processing using teams of mobile agents is presented in Section 3, where we describe the role and contribution of the different classes of agent involved. Section 4 provides a comparison with similar systems. In Section 5 we provide results of experiments we have performed to analyse our system. Finally, Section 6 presents our conclusions.

2.0 Mobile agents

A mobile agent is a software component that can move between hosts within a computer network. To execute remotely, mobile agents move to a machine running a mobile agent server, which provides an interface to the underlying host machine, and facilities to transmit and receive agents. Several mobile agent packages currently provide this functionality, e.g. Voyager [14], AgentTcl [10], Mole [16] and Aglets [6]. The core of an agent server is the virtual machine, onto which mobile agents are loaded and executed. This helps hide from the developer the complexities of moving between, and executing on, differently configured remote hosts. Mobile agent packages also provide facilities for high-level messaging (for inter-agent communication), and high-level methods for controlling agent behaviour, (for instance, moving the agent to a new host). In [12] Lange advocates a convincing case for mobile agent technology, so we shall concentrate on those aspects that would appear to make mobile agents a suitable basis for a DVM, namely:

- mobile agents provide a ready implemented mechanism for the delivery and execution of task code on remote machines;

- as they execute at remote locations mobile agents will be inherently better at assessing the computing resources available; more accurate information would result in more accurate dynamic load balancing;
- most mobile agents package provide a pre-implemented communication mechanism, this could considerably simplify the code used to wrap the task processes;
- once communication with tasks is possible, better management functionality can be built into the manager;
- communication also makes possible inter-agent co-operation, (this could be used to ensure the network resources are efficiently 'farmed' and not over-exploited).

A serious application area for mobile agents which is being investigated [23] is network management, as the above properties are ideally suited to this task. It therefore appears logical to consider using mobile agents as the core management system for an automated DPP system, which also requires similar capabilities

2.2 Mobile Agents for DPP

One possible design approach to using mobile agents for DVM systems would be to create a single generic agent, which would reside at each machine and oversee local resources, (equivalent to the PVM daemon process). This homogenous design is common in current mobile agent applications, [6,10]. This method however has the disadvantage of creating a relatively large and complex agent that needs to handle a large range of functions, i.e.:

- **Navigation** – maintaining an itinerary and knowledge of target sites.
- **Negotiation** – ability to negotiate services and payments with host servers.
- **Security** – protection against malicious attack from host servers or agents.
- **Task Management** – ability to deploy the original task, e.g. a search algorithm.
- **Communication** – ability to establish communication with other agents.
- **User Interface** – the means of interacting with users through a graphical interface.

Whilst these features are desirable they all contribute to increasing the agent's size. If the agent's size becomes substantial, we will not only fail to reduce network load as intended, but may find that more resources are consumed transmitting the agents than are used for task processing. Obviously this eventuality would immediately rule out mobile agents as a viable vector for DPP.

Solving the Security and Resource management problems will also have to be addressed. Indeed, this is an active field of research in its own right, which includes many potential strategies for restricting the access granted to guest processes. For instance, one approach is to treat resources like a financial commodity, charging agents a sum of virtual ‘money’ according to their importance or current supply and demand, [21]. Regardless of what strategy is chosen, the system must incorporate a means of enforcing restrictions, and the ability to apply sanctions if these limits are exceeded.

3. The Team Agent Approach

The remainder of this paper describes a mobile agent based DVM that involves several agent roles, with different functions and skills. As these agents behave co-operatively as a team our system is called **MATS**: the Mobile Agent Team system. One of the design objectives of MATS was to minimise the code that would need to be moved about the network. This was achieved by localising the most complex functionality in an anchored (non-mobile) agent, which can then communicate with the simpler specialised mobile agents through a message passing mechanism. In effect, this is a marriage of the autonomous and social features of collaborative agents with the facilities offered by mobile agent technology.

The resulting mobile agents are lightweight, meaning they require less processor time to be serialised, and are quicker to transmit. This has an important implication for the viability of parallel processing. If the agents are large and cost a significant amount of machine resources to transmit, then the benefits of solving the problem in parallel may be lost.

3.1 The Team Agents

To minimise the complexity of our mobile agents, we have differentiated them according to particular roles. An analogy can be drawn with social insects, which often contain multiple forms within one species, each performing a particular supporting role in the society. MATS consists of the following agents, the relationships between which are shown in Figure 1.

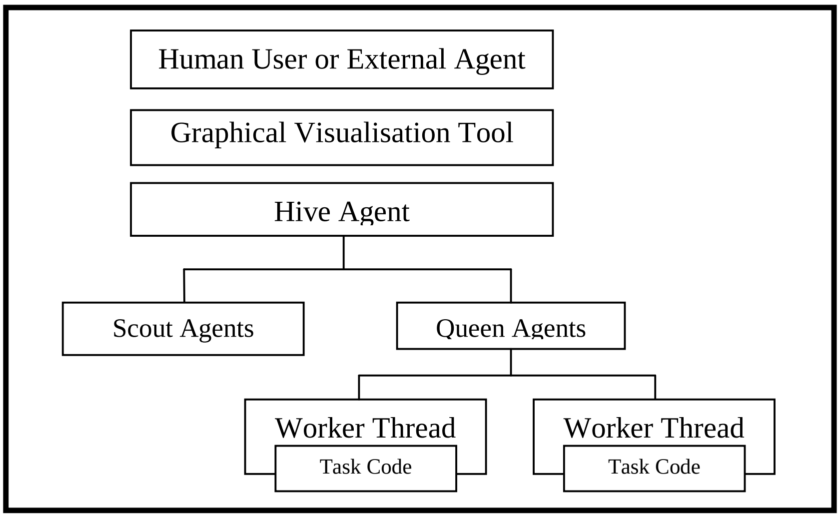

Figure 1 - The Hierarchical relationships between components of the Mobile Agent Team System.

- **Hive** agents – provide the user interface, and co-ordinate teams of mobile agents
- **Queen** agents – team leaders, they monitor hosts, and launch worker threads
- **Scout** agents – these move around the network analysing computer resources

3.2 The Hive Agent

The Hive is an anchored (non-mobile) agent, equivalent to the manager process described in Section 2. It is envisioned that each local area network will possess a single Hive agent, which will be used as a shared resource by the network users. The Hive has the following responsibilities:

Task Wrapping

To begin a distributed computation session the user sends the task code to be processed to the Hive, and specifies the parameter ranges that will be used. The Hive must then decompose the problem into the optimum level of task granularity. This is clearly a difficult requirement and the user may be forced to structure the problem in such a way that can be easily broken down into a set of co-operating tasks. Each component process is then 'wrapped' i.e. associated with the necessary code for mobility, messaging, and distribution.

Resource Assessment and Modelling

The Hive agent is charged with maintaining an accurate model of the local network environment. It is, however, extremely difficult for an anchored agent to accurately assess the status of a remote resource. Hence we have used the specialisation principle to create a lightweight '**Scout'** mobile agent to perform assessment of network resources.

Scout agents are periodically created by the Hive and dispatched to hosts around the network. On arriving at a remote host Scouts perform tests on the local resources and analyse what hardware and software is present. This provides a good example of a task that benefits from being mobilised, as the Scout can evaluate the state of the machine in ways that are not possible remotely. For instance, it can monitor local events to check if the keyboard is being pressed, and record whether the amount of free memory is static, falling or rising. The Scout then moves to the next location on its itinerary or terminates itself.

Infrastructure Maintenance

As all the vectors of the Hive are mobile agents, it can not utilise a remote host unless an agent server is running on it. The Hive will thus attempt to maintain a mobile agent infrastructure in its network neighbourhood. If a machine is detected that is not running an agent server, the Hive will ‘telnet’ into it and run an agent server as a background process. (The mobile agent libraries required being loaded from a logical network drive).

Delegating the creation of agent servers to the Hive not only helps hide details of mobile agent technology from the user, but also makes the initialisation of the DVM much simpler than is the case with systems like PVM. As agent servers are built through telnet calls, servers can be built on any machine for which a user has ‘log-in’ permission. As a result the MATS system obeys the pre-existing network privileges, ensuring the agents do not proliferate into the Intranet space owned by other groups.

Itinerary Planning

The Hive uses its Resource List to plan the itineraries of its mobile agents. The Hive’s distribution rules ensure that Scouts are sent to the last hosts to be visited. Likewise Queens are dispersed as widely as possible so as to maximise the amount of time before a Queen lands on a host with an existing colony.

A key advantage of using a centralised Hive agent is that it can become the steward of its local resources and make management decisions that would be very difficult for a fully distributed society. If, for example, the Hive detects that the available computing resources are approaching saturation, it can recall or destroy its mobile agents, hence preventing “The Tragedy of the Commons”, whereby a shared resource is unintentionally overexploited by different users. This problem would be very difficult to detect if there where several teams acting selfishly and independently. The Hive will also try to optimise itineraries to attempt to minimise the bottlenecks inherent in parallel processing.

Visualisation

Users interact with the MATS system through the Hive’s visual front end. A key feature of this interface is its ability to visualise the network and geographic location of the mobile agents. Visualising mobile agents is inherently difficult as they move between sites and so must be continuously tracked. Furthermore, unlike conventional code, mobile agents are not always available for analysis and interaction, and may even have moved to a site that is not continuously connected to the network.

MATS currently supports the visualisation of three perspectives of the mobile agents: one is the physical layout of the local network, for example, as represented by the floor plan display shown in Figure 2. This representation shows the location of each agent in the environment, (in this case a floor plan). This view also includes information on the loading of each server, with different images depicting the type of agent resident at each location. The side-panels shown in Figure 2 display real-time data received from the mobile agents.

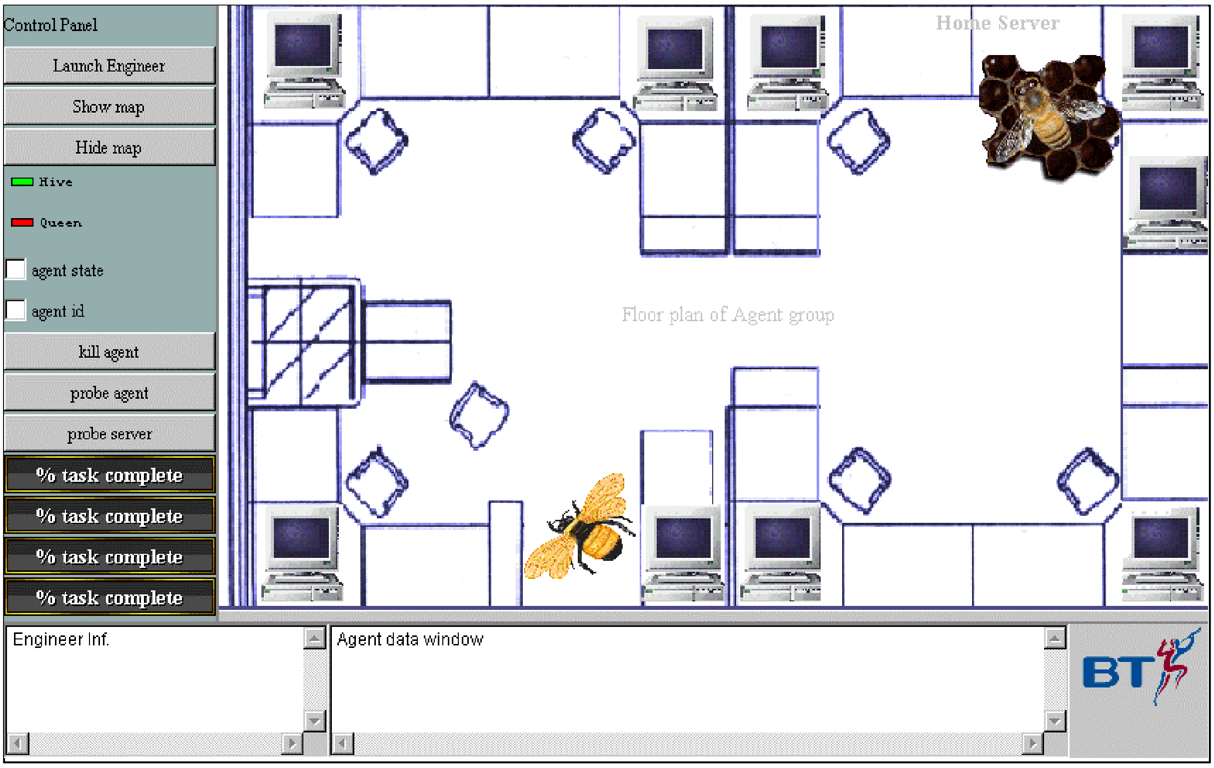

Figure 2 - A screen-shot of the MATS graphical user interface. The main panel shows a floor plan, with the actual positions of available computers superimposed on it. The 'honeycomb' icon at the top-right shows the current position of the Hive, whilst a Queen agent is shown by the bee icon at the bottom. The user interacts with the controls on the left-hand panel, which also shows animated status bars, which are dynamically updated as intermediate results are received from remote Queen agents.

Other visualisation options include a map based geographic representation for use when an agent moves to a remote machine outside the local area network, and a network-orientated display that shows the relative distance between Hive and servers based on ping times.

As well as enabling the user to monitor the remote agents and their current task status, the MATS visualisation tool also provides a means for the user to send instructions to the Hive agent. Typical requests include starting a new task, aborting an existing task, and removing particular machines from the network model (ensuring no mobile agents visit sensitive machines).

Processing Management

Once a distributed computation has been started it is the Hive's duty to monitor its progress, collect and collate results, and if necessary send new instructions to the mobile agents. For instance, if the Hive detects a sub-task is falling behind schedule unexpectedly, it can begin load balancing to derive a new plan that will compensate. This involves constructing a new plan with new itineraries that are sent to the mobile agents, redirecting them to the most appropriate hosts.

To summarise, the Hive is responsible for strategic planning, user interaction and process management. The Hive has no operational role in the parallel processing of tasks though, in our system this is the responsibility of the Queen agents, which are discussed next.

3.3 Queen Agents

A Queen is a mobile agent that is responsible for solving part of the larger problem. The Queen creation process involves Task Wrapping (q.v.), the issuing of operational constraints (like the maximum amount of memory it can use), and provision of an itinerary based on the Hive's resource list. The Queen will then become active and move to the first site on its itinerary. On arrival the Queen analyses the local conditions and launches as many '**worker'** threads as the local machine can comfortably support, the worker threads then begin to solve the task in question.

In the meantime the Queen waits, collecting results from its workers, assigning new tasks and dispatching messages with incremental progress reports to the Hive. This strategy minimises the potential loss of information, since if a Queen is lost, only the sub-task being tackled since the last communication is lost; and the Hive can easily spawn a new Queen to continue the lost sub-task.

The Queen also monitors the local host resources (just as the Scout does) for indications of activity that would suggest that someone has begun to use the machine again. In this event, the Queen will kill its worker threads and leave for the next machine in its itinerary. As our intention is only to utilise spare computing resources, not compete for resources with the legitimate owners of each machine, it is important that the mobile agents are as unobtrusive as possible. Alternatively, the Queen may act as a nice process and simply freeze its current activity until the current machine is free again. The choice would depend on what the Hive knows of that machine's pattern of activity.

Worker Threads

The worker threads are not mobile agents, but processes created by the Queen to perform part of some larger computation. Each thread is an instantiation of the user-supplied problem task code given to the Hive at the start of the process. When threads are launched the Queen provides parameters appropriate to the problem in hand, with new parameters provided when the thread completes its allocated task.

To ensure the worker threads can not outlive their usefulness, each worker thread is created with an in-built destruct mechanism that makes it dependent on the Queen for its survival – so even if the Queen dies, the worker will soon kill itself. Queen agents also have limited (albeit much longer) life spans, and must remain in periodic contact with the Hive to survive.

Resource Management

As mobile agents possess the same functionality as viruses, concern has been expressed about mobile agents proliferating out of control, consuming all available resources. In MATS, reproductive ability is restricted to the Hive and the Queen agents. This should minimise the risk of MATS being used for ’denial of service’ attacks by malicious users, as their task code is not permitted to launch new threads. Queens also have a regulatory role monitoring local machine loading, and killing any task using too many resources, a capability that can not be overridden by users.

Population Control

To prevent accidental proliferation, MATS incorporates a ‘chain of command’ from the Hive to the individual Worker threads. Thus if the Hive terminates a particular Queen agent, its dependent workers will quickly die in the absence of the periodic maintenance signals. This feature is implemented by most mobile agent packages, and is analogous to apoptosis in organic systems: the programmed cell death mechanism that prevents cancer. Likewise, if a Queen is 'forgotten' by the Hive, it too will soon expire and terminate.

4.0 Related Systems

As mentioned earlier, two recent developments in the field of distributed computation are SUN’s Jini [22] project, and Microsoft’s Millennium project, [13]. These projects aim to provide a new level of abstraction for application programmers and users, managing machines and network connections for the programmer in a self-configuring manner. The MATS system by contrast is a higher level application, and is thus best compared with other DVM systems.

4.1 The PVM System

One aspect of the MATS system that can be compared against PVM is the process of task allocation. In the MATS system, the corresponding structure is the Worker thread, which takes advantage of the multi-threading ability of the Java language. However it is also targeted at course grain application decomposition. The major difference is the ability of the worker threads to move across the network in response to requests from the parent Queen agents. In both cases it is the user’s responsibility to deconstruct the required application into an appropriate form for parallel operation. Fortunately, the use of Java in MATS eliminates many problems associated with machine specific data structures.

Comparing resource management with PVM, an advantage of the MATS approach is the automatic collection of global resource data, which flows from the Queen agents or scouts up to the Hive agent. This is necessary within a mobile code system, if each Queen had global knowledge then they may engage in race conditions to land on machines with the best resources. In contrast, with a static system like PVM it is possible for the support processes to have global knowledge. The Hive agent can therefore offer a user level view of the current virtual machine state.

4.2 The Harness System

Recent work by the PVM group has focused on creating a more dynamic system, termed HARNESS [2,5], with a Java kernel, which adopts a ‘pluggable’ approach. This allows the required dynamic merging of multiple DVMs and adaptive reconfiguration, (used primarily for co-operative computing). This is an attempt to solve what are perceived as the several inherent problems with existing PVM type systems, an inability to fully utilise heterogeneous platforms and an inability to create migrating or co-operative applications.

However, linking separate DVMs is a non-trivial administrative operation, as state information must be shared, and protocols agreed. The intention is to create a more flexible environment that allows for multiple-user co-operation with large scale computing applications. The MATS system has been designed to support wide-scale co-operation, using the Voyager class library’s *Space* system for multi-cast messaging. This would allow the Hives of different MATSs to co-operatively work together by broadcasting messages-events into each other’s sub-spaces, (explained in [14]).

5. Example: Parallel Processing with Team Agents

The MATS system has been tested on a distributed Genetic Programming (GP) problem, i.e. symbolic regression of a curve [11]. Copies of the GP problem were assigned to several Queen agents, which were distributed across the computing resources of a local area network, (comprising several UltraSparcs and dual processor PCs running Windows NT).

The first test application of the system was for a distributed Genetic Algorithm problem, which was then replaced with the parallel GP application. This was a very simple process with minimal re-coding, which validated one of the main goals of achieving a user-friendly system.

In both cases the system facilitated the rapid deployment and management of the task code, and allowed useful visualisation of the real-time operation of the system.

The asynchronous and autonomous nature of mobile agents provided a fault tolerant and robust system. Tests demonstrated that failure of a single host or even temporary failure of the Hive agent did not seriously disrupt the system, as each remaining Queen agent continues to operate autonomously until contact with the Hive agent is re-established.

MATS is implemented in the Java programming language, using the Voyager mobile agent platform from ObjectSpace Inc. [14]. Voyager was chosen because it provides a well-developed mobile agent class library, that is also stable and well documented. The Voyager package uses the 1.1 release of the Java language, as it supports object serialisation.

5.2 Future Work

At the infrastructure level, improved message/event handling is required to reduce the machine load caused by Java threads, especially where synchronisation locks are used. Poor design of the synchronisation methods, (used to prevent deadlocks), is a serious potential drain on CPU resources. At the application level, learning by the Hive agents would be a significant area for development, which would allow greater autonomy and improved load balancing. If these agents could use acquired knowledge of the activity patterns within the network, then they could pre-emptively allocate tasks in an optimum manner.

For some high performance computations the impact of using Java rather than a faster, optimised language like C may become significant. However, it is possible to reduce the performance gap by compiling Java to native code, linking to native code libraries, or by using a Just-In-Time compiler. We feel that performance overhead of using Java is more than off-set by the higher level functionality it provides, e.g. networking ability and true portability. Interestingly, the HARNESS project has adopted a Java core for the task management, and a Java port of the PVM architecture, [19] gives specific figures for comparing the message handling performance between JavaPVM and PVM.

6.0 Conclusions

This work offers a novel system for distributed parallel processing, operating over heterogeneous environments, with dynamic reconfiguration of the management system. This has been motivated by our desire to simplify the configuration and usage of a distributed virtual machine and to use the principle of agent co-operation to improve a mobile agent application. We hope that this work has demonstrated that an agent-based solution provides a viable, and often better, alternative to current static, non-mobile DVMs.

Whilst mobile agents would seem to be an obvious vector for distributing and performing parallel computations, the extra layer of abstraction imposes many overheads. Consequently we developed specialised entities, each performing particular roles, which act together as a *team* to achieve user defined goals. By using a two stage management system we have incorporated strategic central planning and a tactical level of operational management.

References

[1] Clements P. E., Papaioannou T., & Edwards J.M., “Aglets: Enabling the Virtual Enterprise”, pub. in ME-SELA `97, available on-line from: http://www.trl.ibm.co.jp/aglets/index.html

[2] Dongarra, Geist, Kohl, Papadopoulos, Scott, & Sunderam. “HARNESS: Heterogeneous Adaptable Reconfigurable Networked Systems”, web based paper, http://www.epm.ornl.gov/harness/.

[3] Finin T., Labrou Y., & Mayfield J., “KQML as an agent communication language”, in Jeff Bradshaw (Ed.), ``Software Agents'', MIT Press, Cambridge, 1997.

[4] “Foundation for Intelligent Physical Agents”, http://drogo.cselt.it/fipa/

[5] Geist, Kohl, Papadopoulos, & Scott “Beyond PVM 3.4: What We've Learned, What's Next, and Why”, web based paper, http://www.epm.ornl.gov/harness/

[6] Harrison G. C., Chess D. M., & Kershenbaum A., “Mobile Agents: Are they a good idea?”, IBM Internal Research Report, T. J. Watson Research Center, 1995.

[7] Geist A., et al. “PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing”, MIT Press Scientific and Engineering Computation, Janusz Kowalik, Editor, Massachusetts Institute of Technology, 1994.

[8] Goldberg D., Kargupta H., Horn J., and Cantu E. (24 pp.) “Critical Deme Size for Serial and Parallel Genetic Algorithms”, Paz IlliGAL Report No 95002*
Available from: ftp://GAL4.GE.UIUC.EDU/pub/papers/IlliGALs/ 95002.ps.Z

[9] Juille H., and Pollack J.B., "Parallel Genetic Programming and Fine-Grained {SIMD} Architecture", (book) Working Notes for the AAAI Symposium on Genetic Programming, 1995, ed. E. V. Siegel and J. R. Koza, pp31-37, MIT press Cambridge.

[10] Kotz D. et al., “Agent TCL: Targeting the needs of Mobile Computers”, IEEE Computing Online: http://computer.org/internet/icl1997/w4toc.htm, 1997.

[11] Koza J., “Genetic Programming: On the Programming of Computers by Means of Natural Selection.” (book) Publisher MIT Press, 1992.

[12] Lange D., “Mobile Agents: Environments, Technologies, and Applications”, Proceedings of PAAM'98, March 1998, pp11-14.

[13] Microsoft Corporation, Operating Systems Directions for the next Millennium, position paper from:

http://www/research.microsoft.com/research/os/Millenium/mgoals.html.

[14] “ObjectSpace Voyager and Agent Platforms Comparison”, public white-paper from http://www.objectspace.com/Voyager/voyager.html.

[15] Papadopoulos P. M., Kohl J. A., Semeraro B. D., ``CUMULVS: Extending a Generic Steering and Visualization Middleware for Application Fault-Tolerance," Proceedings of the 31st Hawaii International Conference on System Sciences (HICSS-31), The Big Island of Hawaii, January 1998.

[16] Strasser M., Baumann J., & Hohl F., “Mole – A Java based Mobile Agent System”, Proceedings of the 2nd ECOOP Workshop on Mobile Object Systems, Austria, 1996.

[17] Sunderam V.S., “PVM: A Framework for Parallel Distributed Computing”, V. S. Concurrency: Practice and Experience, 2, 4, pp 315—339, December 1990. Available from: http://www.netlib.org/ncwn/pvmsystem.ps

[18] White, J. E., Helgeson, C. S., and Steedman, D. A. “System and method for distributed computation based upon the movement, execution, and interaction of processes in a network”. General Magic, US Patent 5,603,031, filed 8 July 1993, issued 11 February 1997.

[19] Yalamanchilli N., & Cohen W., “Communication Performance of Java based Parallel Virtual Machines”, submitted to Java 1998 conference. http://www.cs.virginia.edu/~ajf2j/jpvm.html.

[20] Home page of distributed computing group, http://www.distributed.net/

[21] Waldspurger C.A., Hogg T., Huberman B.A., Kephart J.O., Stornetta W.S. “Spawn: a Distributed Computational Economy”. IEEE Transactions on Software Engineering, 18(2), p103-117, Feb 1992.

[22] Sun Microsystems, “What Is Jini?” http://www.java.sun.com/products/jini/whitepapers/whatisjini.pdf

[23] Minar N., Kramer K., & Maes P. “Cooperating Mobile Agents for Mapping Networks”, Submitted to Cooperative Information Agents 1998, available from, http://lcs.www.media.mit.edu/people/nelson/research/routes-coopagents/