Performance Evaluation of Mobile Agents: Issues and Approaches

Marios D. Dikaiakos and George Samaras

Department of Computer Science
University of Cyprus, CY-1678 Nicosia, Cyprus
{mdd,cssamara}@ucy.ac.cy

1 Introduction

With the emergence of Internet as a world-wide infrastructure for communication and information exchange, Internet-based distributed applications have gained remarkable popularity. One of the most promising approaches for developing such applications is the Java-based mobile-agent paradigm [5,8,15,39]. Mobile Agents (MA) are being used already in a variety of applications ranging from telecommunications management to Web databases, cooperative environments information-gathering systems, electronic commerce systems, intelligent distributed systems, and so on [23,24,7,9,4]. In that context, a distributed application can be considered as a dynamic group of agents working in coordination to accomplish some goal.

Mobile agents offer a number of diverse advantages in the development of distributed systems including enhanced programmability through modular, object-oriented structures and the increased flexibility provided by mobility; performance optimization for distributed operations that involve heavy network delays and/or weak connectivity; extended autonomy in terms of existing support for asynchronous execution and disconnected operations [6,8,21]. The employment of MA technologies for the development of next- generation Internet systems opens numerous research problems related to programming APIs and tools, security, fault-tolerance, design paradigms and programming techniques, communication, intelligence, scalability and performance [18].

The issue of performance, in particular, is very important in emerging Internet-systems: numerous studies show that performance of systems and applications determines to a large extent the popularity of Internet services and user-perceived Quality of Service [1,3]. Moreover, performance evaluation is crucial for performance "debugging," that is the thorough understanding of performance behavior of systems. Results from performance analyses can enhance the discovery of performance and scalability bottlenecks, the quantitative comparison of different platforms and systems, the optimization of application designs, and the extrapolation of properties of future systems.

So far, systematic performance studies and experiments have provided important insights for the design of parallel and distributed systems and the tuning of applications [13,14,17,29]. Nevertheless, the more complex a system or application is, the harder its evaluation becomes, because of the large variety of factors that affect its performance [10]. This observation is particularly true for systems and applications running on top of gigantic platforms like Internet [1]. The performance evaluation of mobile-agent systems is even harder than the analysis of more traditional parallel and distributed systems, due to the dynamic nature and agile configuration of mobile agents.

The objective of this chapter is to investigate issues pertinent to performance engineering for mobile-agent platforms and systems. First, we describe briefly the basic characteristics of mobile agents. Then, we explore the notion of performance evaluation in the context of mobile agents and present an overview of recent approaches for performance evaluation and analysis of mobile-agent systems. These approaches include benchmarking efforts, scalability studies, analytical models, and Petri-net modeling.

Finally, we present a novel performance analysis approach that we developed to gauge quantitatively the performance characteristics of different mobile-agent platforms [12,29]. We materialize this approach as a hierarchical framework of benchmarks designed to isolate performance properties of interest, at different levels of detail [12,29]. We identify the structure and parameters of benchmarks and propose metrics that can be used to capture their properties. We present a set of proposed benchmarks and examine their behavior when implemented with commercial, Java-based, mobile-agent platforms.

2 Mobile Agents

In mobile applications, data may be organized as collections of objects, in which case objects become the unit of information exchange between mobile and static hosts. Objects encapsulate not only pure data but also information regarding their manipulation, such as operations for accessing them. Incorporating active computations with objects and making them mobile leads to *Mobile Agents*.

Mobile agents are processes dispatched from a source computer to accomplish a specified task [21,35,38,39]. Each mobile agent is a computation along with its own data and execution state. In this sense, the mobile agent paradigm extends the RPC communication mechanism, according to which a message is just a procedure call whereas now it is an object with state and functionality. After its submission, the mobile agent proceeds autonomously and independently of the sending client. When the agent reaches a server, it is delivered to an agent execution environment. Then, if the agent possesses necessary authentication credentials, its executable parts are started. To accomplish its task, the mobile agent can transport itself to another server, spawn new agents, or interact with other agents. Upon completion, the mobile agent delivers the results to the sending client or to another server.

By letting mobile hosts submit agents, the burden of computation is shifted from the resource-poor mobile hosts to the fixed network. Mobility is inherent in the model; mobile agents migrate not only to find the required resources but also to follow mobile clients. Finally, mobile agents provide the flexibility to adaptively shift load to and from a mobile host depending on bandwidth and other available resources. Mobile-agent technology is suitable for wireless or dial-up environments [21,25].

Mobile agents are typically written in interpreted languages, such as Java and Tcl, and thus tend to be independent of the operating system and hardware architecture. A number of commercial and research projects have developed mobile-agent environments, such as Aglets by IBM [8], Concordia by Mitsubishi [40], Voyager by Objectspace [15], Grasshopper by IKV++ [5], Mole by the University of Stuttgart [33], etc.

3 Different Approaches for MA Performance Analysis

Traditionally, the performance assessment of software systems is conducted through experimentation and monitoring, simulation, modeling and combinations thereof. The more complex a system is the harder its performance evaluation becomes, dictating the employment of these techniques at various levels of abstraction. To this end, software systems are modeled as hierarchical structures of interacting modules, i.e., subsystems and objects; each module is assigned a performance model that incorporates performance and load parameters of relevance, and a description of the underlying architecture and workload [41]. Model development is performed in a "top-down" manner, starting from high-level structure and moving towards code implementation. Experimentation and/or simulation can be used at various layers of abstraction to specify the values of modeling parameters.

The development and assembly of performance models for MA systems is more complicated than for more "traditional" parallel, distributed or object-oriented software; when analyzing the performance of MA-based systems, we must take into account issues, such as:

– The absence of global time, control and state information: this makes it hard to define and determine unequivocally the condition of a particular MA-system at a particular moment.
– The complex architecture of MA platforms: simple models and metrics used for the performance characterization of typical parallel and distributed systems are not adequate for isolating performance problems of MA systems. Further investigation and definition of more complex models and metrics are necessary.
– The variety of distributed computing (software) models that are applicable to mobile-agent applications: this variety dictates the design of different experiments, tailored to the different software models of interest.
– Construction of simple and portable benchmarks for experimentation is difficult due to the diversity of operations found in MA platforms.
– The presence of mobility, which makes it hard to establish a concise and stable representation of system resources that affect MA performance, due to the dynamic nature and agile configuration of MA systems.
– The additional complexity introduced by issues that affect the performance of Java, such as interpretation versus compilation, garbage collection, etc.

A number of recent projects have addressed the issues mentioned above. One thread of work examines the relative advantages of the mobile-agent paradigm versus other distributed-computing models from a performance perspective:

For instance, in [34], Strasser and Schwehm introduce a mathematical model to compare analytically the performance of *agent migration* and *remote execution* with the more traditional approach of *remote procedure calls* (RPC), in the case where mobile agents are used for filtering information in information retrieval applications. Their performance model takes into account issues, such as network throughput, communication latency, and network load. Furthermore, the size and execution time for RPCs, the size of agent code, data, and state, the "selectivity" (filtering ratio) of an agent, etc. The authors use their model to identify situations where agent migration has performance advantages over remote procedure calls. Analytical results are corroborated with experimentation using the Mole platform [33].

A similar problem is studied by Puliafito, Riccobene, and Scarpa in [26]. The authors compare the mobile-agent, remote-evaluation and client-server models of distributed computing. To this end, they use non-Markovian Petri nets modeling, applying probability distributions to model parameters, such as request size, time for searching data in a server, processing time, size of replies and queries, code size for migrating codes, and throughput of the communication network. Petri-net analysis shows that for the scenarios examined, which are pertinent to information filtering applications, the performance advantage of mobile agents arises under certain "external" to this model factors, such as network connectivity and speed.

Mathematical modeling is used by Kotz, Jiang, Gray, Cybenko and Peterson to study a more extended mobile-agent application scenario in [20]. According to this scenario, mobile agents are used to support a data-filtering application involving many wireless clients that filter information from a large data stream arriving across a wired network from a server. The mathematical model is used to compare analytically two alternative approaches: a) The server combines and broadcasts all the data streams over the wireless channel and filtering takes place at each client site. b) Each client dispatches an agent to the server; the agent monitors and filters the data stream before sending relevant data to its corresponding client. The authors use two performance metrics for their comparison study: computation and bandwidth requirements. They conclude that the mobile agent approach trades server computation and cost for savings in network bandwidth and client computation, which is an important remark in the context of "thin" clients used in mobile-computing applications.

In addition to mathematical analysis and Petri-net modeling, there has been a range of simulation and experimental studies on mobile-agent systems. In [31], Spalink, Hartman and Gibson study the performance advantages of employing a mobile agent for conducting search within a file at the file-server's location instead of searching remotely over the network. Trace-driven simulation shows that the MA approach is advantageous when the server's CPU is not a bottleneck.

Another thread of work employs Petri-net modeling and experimentation to investigate performance properties of mobile-agent systems: For instance, General Stochastic Petri nets are used by Rana in [27] to investigate the performance properties of two agent design patterns [2], "Task" and "Interaction," and a combination thereof. The study is performed in the context of an agent-based, e-commerce application. The Petri-net models introduced are executed with a Petri-net simulator to study the performance and scalability of the underlying application. Furthermore, Petri nets are used by Rana and Stout in [28] to model the performance of multi-agent systems in a way that captures properties arising both from agent-collaboration requirements pertinent to multi-agent applications and the performance characteristics of MA systems.

Samaras, Dikaiakos, Spyrou and Liverdos in [29] propose, employ, and validate an approach to evaluate and analyze MA performance in the context of basic mobile-computing models applied in the provision of distributed database access over the Web. The proposed experimental setup is used to compare quantitatively two commercial MA platforms and to study the performance of different approaches for database access over the Web, using mobile agents. In [11,12], we extend our previous work and propose a hierarchical framework that can be used to investigate quantitatively and experimentally various aspects of mobile-agent performance. This framework is implemented as a set of benchmarks and used to study the performance and scalability of a number of commercial MA platforms. More details are given in subsequent sections. In a similar vein, Silva, Soares, Martins, Batista and Santos define and run benchmarks in [30], to evaluate the performance of some of the existing mobile agent platforms.

4 A Framework for Investigating MA Performance

To cope with the complexities of MA performance analysis we propose the adoption of a hierarchical approach inspired by the structure of MA-based applications. This structure is determined by:

1. The MA platform adopted to program a particular application. Mobile-agent platforms (such as [4,5,15,19,39]) are systems that provide some basic functionality supporting the mobility of objects (transportation and location services), the communication between them, security, fault-tolerance etc. Access to this functionality is granted through a platform-specific programming interface provided for the development of applications. Various MA platforms differ in terms of their functionality, programming interface and performance characteristics which are dictated by underlying implementation details.
2. The high-level abstractions representing software-design choices made by developers for a particular application. These abstractions are implemented with the programming interface of the MA platform employed.

Notably, the differences of mobile-agent platforms and the variety of application domains result to a huge space of options vis-a-vis MA-system structure. We abstract most of these options away by focusing on *Basic Elements* and *Application Kernels*.

4.1 Basic Elements and Application Kernels

We define as *Basic Elements* of mobile-agent platforms a set of basic abstractions that incorporate the fundamental functionalities commonly found and used in MA platforms. For the objectives of our work, the Basic Elements of MA platforms are identified from existing, "popular" implementations as follows [4,5,15, 19,39]:

– *Agents*, defined by their state, implementation (bytecode), capability of interaction with other agents/programs (interface), and a unique identifier.
– *Places*, representing the environment in which agents are created and executed. A place is characterized by the virtual machine executing the agent's bytecode (the *engine*), its network address (location), its computing resources, and any services it may host (e.g., a database gateway or a Web-search program).
– *Behaviors* of agents within and between places, which correspond to the basic functionalities of a MA platform: creating an agent at a local or remote place, dispatching an agent from one place to another, receiving an agent that arrives at some place, communicating information between agents via messages, multicasts, or messenger agents, synchronizing the processing of two agents, etc.

Basic elements of MA systems are combined into scenarios of MA-use, which we call *Application Kernels*. Application Kernels define solutions common to various problems of agent design and are defined in terms of places participating in a scenario, agents placed at or moving between these places, and interactions of agents and places (agent movements, communication, synchronization, resource use). Application Kernels correspond to widely applicable models of distributed computation on particular application domains [32], and represent widely accepted and portable approaches for addressing typical agent-design problems [2]. Typically, Application Kernels are the building blocks of larger MA applications.

Consequently, we define Application Kernels that correspond to the Client-Server (**C/S**) model of distributed computing and its extensions for mobile computing: the Client-Agent-Server model (**C/A/S**), the Client-Intercept-Server model (**C/I/S**), the Proxy-Server model, and variations thereof that use mobile agents for communication between the client and the server (**C/S-MA**, **C/A/S-MA**, **C/I/S-MA**; see Figures 1 and 2). More details on these models are given in [29,32].

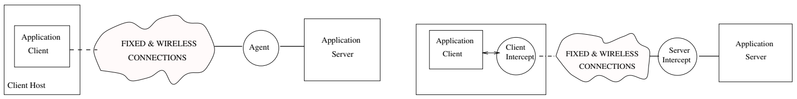


Fig. 1. The Client-Agent-Server and Client-Intercept-Server Models.

Additional Application Kernels correspond to the *Forwarding* and the *Meeting* agent-design patterns, defined in [2,8]. The *Forwarding* pattern "allows a given host to mechanically forward all or specific agents to another host" [2]. The *Meeting* pattern provides a way for two or more agents to initiate local interaction at a given host (see Figure 2) [2,37]. We chose the *Forwarding* and *Meeting* patterns because they can help us quantify the performance traits of agents and places in terms of their capability to re-route agents and to host inter-agent interactions.

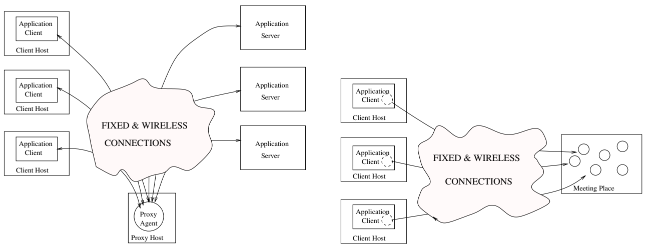


Fig. 2. The Proxy-Server Model and the Meeting Pattern.

4.2 A Hierarchical Performance Analysis Approach

To analyze the performance of mobile-agent applications, we need first to develop an approach for capturing basic performance properties of MA platforms. These properties must be defined independently of how particular mobile-agent API's are used to program and deploy applications and systems on Internet. To this end, our approach focuses on Basic Elements of MA platforms and seeks to expose the performance behavior of these functionalities: how fast they are, what is their overhead, if they become a performance bottleneck when used extensively, etc.

Having isolated the performance characteristics of basic MA elements, we focus on performance traits of Application Kernels in order to explain the performance behavior of full-blown applications, which use these kernels as building blocks. Performance traits of an Application Kernel depend on the characteristics of its constituent (basic) elements, and on how these elements are combined together and influence each other. To identify how a kernel affects application performance, we have first to isolate its basic performance properties, that is metrics capturing its performance capacity, overheads incurred by the interaction of its constituent elements, bottlenecks affecting kernel-performance, etc. For example, an Application Kernel could involve an agent residing at a place on a fixed network and providing database-connectivity services to agents arriving from remote places over wireless connections. This kernel may exist within a large digital library or e-commerce application. It may, as well, belong to the "critical path" that determines end-to-end performance of that application. To identify how this kernel affects overall performance, we have to find out what is the overhead of transporting an agent from a remote place to a database-enabled place, connecting to the database, performing a simple query, and returning the results over a wireless connection. Interaction with the database is kept minimal because we are trying to capture the overhead of this kernel and not to investigate database behavior.

Performance analysis of Application Kernels involves the use of simple workloads seeking to discover fundamental performance properties. It is very interesting, however, to explore the performance behavior of instances of these kernels under conditions expected to occur in a real execution of a full-blown application. To this end, we can enrich the scenarios implemented by Application Kernels by extending the functionality of mobile agents and by simulating realistic workload conditions.

In view of the above remarks we propose a framework for the Hierarchical Analysis of MA-performance. Our framework consists of four layers of abstraction:

1. At a first layer, it explores and characterizes performance traits of Basic Elements of MA platforms.
2. At a second layer, it investigates implementations for popular Application Kernels upon simple workloads.
3. At a third layer, it studies Micro-Applications, that is, implementations of application kernels which realize particular functionalities of interest (e.g., database connectivity), running on realistic workloads.
4. Last but not least, at a fourth layer, our framework studies full-blown Applications running under real conditions and workloads.

Our approach has to be accompanied by proper metrics, which may differ from layer to layer, and parameters representing the particular context of each study, i.e., the processing and communication resources available and the workload applied. It should be stressed that the design of our performance analyses in each layer of our conceptual hierarchy should provide measurements and observations that can help us establish causality relationships between the conclusions from one layer of abstraction to the observations at the next layer in our performance analysis hierarchy.

4.3 Parameterizing Basic Elements and Application Kernels

To proceed with performance experiments, measurements and analyses, after the identification of Basic Elements and Application Kernels, we need to specify the *parameters* that define the context of our experimentation, and the *metrics* measured. Parameters determine the *workload* that drives a particular experiment, expressed as the number of invocations of some Basic Element or Application Kernel; large numbers of invocations correspond to intensive use of the element or kernel during periods of high load. Furthermore, the *resources* attached to participating places and agents: the channels connecting places, the operating system and hardware resources of each place, and the functionality of agents and places.

The exact definition of parameters and parameter-values depend on the particular aspects under investigation. For example, to capture the intrinsic performance properties of Basic Elements, we consider agents with limited functionality and interface, which carry the minimum amount of code and data needed for their basic behaviors. These agents run within places, which are free of additional processing load from other applications. Places may correspond either to agent servers with full agent-handling functionality or to agent-enabled applets. The latter option addresses situations where agents interact with client-applications, which can be downloaded and executed in a Web browser. Participating places may belong to the same local-area network, to different local-area networks within a wide-area network, or to partly-wireless networks. Different operating systems can be considered.

Parameters become more complicated when studying application kernels. For instance, when exploring Client-Server types of models, we have to define the resources to be incorporated at the place which corresponds to the server-side of the model. Resources could range from a minimalistic program acknowledging the receipt of an incoming request, to a server with full database capabilities.

5 Implementation and Experimentation with Benchmarks

The hierarchical performance analysis framework presented in the previous section can be applied to study performance characteristics of different MA platforms and investigate MA-based applications. To this end, we propose three layers of benchmarks that correspond to the first three layers of the hierarchy presented in the previous Section. These benchmarks are defined as follows:

– **Micro-benchmarks:** short loops designed to isolate and measure performance properties of basic behaviors of MA systems, for typical system configurations.
– **Micro-kernels:** short, synthetic codes designed to measure and investigate the properties of Application Kernels, for typical applications and system configurations.
– **Micro-Applications:** instantiations of Application Kernels for real applications. Here, we involve places with full application functionality and employ realistic workloads complying to the *TPC-W* specification [36].

5.1 Experimentation

For our experiments, we focus on a set of micro-benchmarks, and on micro-kernels corresponding to database access over the Web. We implement these benchmarks with two commercial MA platforms: IBM's Aglets and Mitsubishi's

Concordia. Before presenting experimental results, we describe the basic characteristics of these two platforms. Further experimental results and comparisons with other MA platforms can be found in [12,29].

The **Aglets Software Development Kit (ASDK)** is an environment for programming mobile Internet agents (Aglets) in the Java programming language [16]. An Aglet is a Java object that has the ability to move (be dispatched) autonomously from one computer host to another. This transportation is possible between hosts with a pre-installed Tahiti server (a "place" in our terminology), which is an Aglet server program implemented in Java. Tahiti captures arriving Aglets and provides them with an Aglet context. In this context, Aglets can run their code, communicate with other Aglets, collect local information and move to other hosts. The protocol used for the transportation of the Aglet, is the Aglet Transfer Protocol (ATP) by IBM, [22]. ATP defines four standard request methods for agent services, which are `dispatch, retract, fetch` and `message`. The Tahiti server (we use v1.0.3 with size 2.25MB) can be installed at any platform that supports Java Virtual Machine (JVM). Fiji Applet is an abstract applet class of a Java package called "Fiji Kit", which allows Aglets to be fired from applets. For a Java-enabled Web browser (like Netscape Communicator) to host and fire Aglets, and thus become a "place," two more components are required and are provided by IBM: an Aglet (fiji) plug-in allowing the browser to host Aglets, and an Aglet router that must be installed at the Web server. The Aglet router's purpose is to capture incoming Aglets and forward them to their destination.

Concordia is a framework for the development, execution and management of mobile agent applications written in Java [19,40]. The Concordia system is made up of several integrated components. The Concordia server (a "place" in our terminology) is the major block, inside which the various Concordia Managers reside. One of these managers is the Agent Manager, which provides the communication infrastructure that enables agents to be transmitted from and received by nodes on the network, and the management of the life-cycle of the agent. Communication in Concordia relies on the Java RMI system. One of the central features of RMI is its ability to download the bytecode of an object's class if the class is not defined in the receiver's virtual machine. To ensure security of all its transmissions, Concordia uses the SSLv3 (Secure Socket Layer) protocol to transmit agent information from one system to another. Concordia provides an abstract class called `ConcordiaApplet` that extends Java's applet class. There is no need to install the Concordia System on a client machine because `ConcordiaApplet` implements a class provided by Concordia, namely `AgentTransporter` that acts as a lightweight Concordia Server. Concordia implements inter-agent messaging through the concept of events, which are Java objects posted at the Event Manager of a Concordia Server. Concordia supports *Service Bridges*, that is, gateways between Concordia and Java resources installed locally at the host machine. An agent can call methods of a Service Bridge and get back its results. Services offered by Service Bridges can be registered with a directory service running in one or more Concordia Servers. Experiments presented below were done with version 1.1.2 of Concordia.

5.2 Micro-Benchmarks

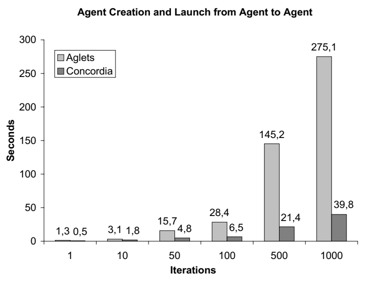
As described in Section 4, computing models that use MA technology employ the following basic components: a) *mobile agents* to materialize modules of the client-server model and its variations; b) *messenger agents* as an approach for flexible communication; c) *messages* as an efficient communication and synchronization mechanism. Therefore, micro-benchmarks devised to test basic performance properties of mobile-agent platforms must focus on measuring the performance of frequently executed components, i.e., messenger agents and messages. To this end, we propose the following benchmarks that measure:

– [**AC/L**]: The overhead of creating and launching messenger agents, which is represented by the time to create and dispatch mobile agents with minimal content.
– [**MSG**]: The overhead of messaging, that is, the time to create and post messages.
– [**ROAM**]: The overhead of agent traveling, which is represented by the time it takes an agent with minimal content to return to its host node after roaming along a given itinerary of hosts. The agent has minimal interaction with the resources of each host visited, e.g., it just queries the host's identification.
– [**SYNC**]: The synchronization overhead, which is represented by the time to exchange a message between two hosts (equivalent to the "ping-pong" benchmark [14]).

Benchmarks are parameterized by the number of iterations they execute. We measure the *total time to completion* of each benchmark for a chosen number of iterations. In our tests, we choose iteration numbers from 1 to 1000.

Micro-Benchmark Experiments: For the quantitative comparison we ran several tests implementing the micro-benchmarks presented above. In our tests, we examine two scenarios: the first scenario presumes the installation of the full agent-execution environment on the hosts of the system under scrutiny. The second scenario tests the case where the client has limited computing resources, as in the case of mobile-computing units, or connects from a machine with minimal configuration, i.e., Internet connectivity and a Java-enabled Web browser. In the second scenario, the client is communicating with the mobile-agent platform by downloading an applet enhanced with agent-handling capabilities (Fiji or Concordia applet).

Test 1 corresponds to benchmark [**AC/L**]. For this test we launch and dispatch agents from a parked agent to a remote Agent Server. We measure the time it takes to create and launch the agents. For both platforms, we employ agents (Aglets and Concordia agents) of identical, minimal, functionality. The size of the Aglet is 1.64 KB whereas the size of the Concordia agent used is 693 bytes. The parked Aglet is 3.54 KB and the parked Concordia Agent is 1.65 KB.



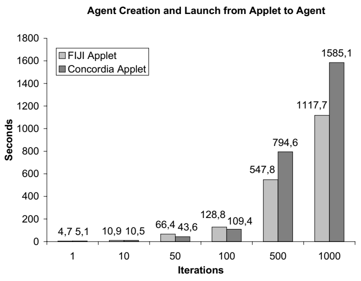


Fig. 3. [AC/L] Benchmark; results from Tests 1 and 2.

Test 2 corresponds to benchmark [**AC/L**] under the second scenario, where we use an applet to launch agents. The Fiji Applet is 4.8 KB in size and the Concordia Applet is 3.61 KB.
Test 3 corresponds to benchmark [**MSG**]. A parked Aglet/Concordia agent creates and sends/posts messages/events to a remote Aglet or Event Manager respectively. The message/event carries a simple piece of information - an integer expressing its ID. The size of the message is 5 KB whereas the size of the event is only 286 bytes.
Test 4 corresponds to benchmark [**MSG**] following the second scenario, where messages are created and launched from an applet. The applet sizes are 5.19 KB for Fiji and 3.09 KB for Concordia.
Test 5 corresponds to benchmark [**ROAM**] with one hop. An agent launches another agent to a remote Agent Server. At its arrival, the agent prints its id and returns back. Upon return to the sender, the agent is re-dispatched towards the same destination.
Test 6 also corresponds to benchmark [**ROAM**] with one hop, where applets act as agent launchers.
Test 7 corresponds to benchmark [**SYNC**], with message exchange taking place between two agents. It is a variation of Test 5 using messages: a message (or event) is sent to a remote receiver. Once this message is received, the receiver sends it back. The next message is sent after the return of the previous one.
Test 8 corresponds to benchmark [**SYNC**] with message exchange occurring between an applet and an agent.

For the above tests we used a Pentium PC at 166MHz with 32MB of RAM running Microsoft Windows 95 as the PC from where we launched the agents and the messages and a Pentium Pro at 350 MHz with 64MB of RAM running Microsoft Windows 95. These computers were connected on a 10 Mbps Ethernet LAN.

Discussion: Our micro-benchmark tests provide useful insights into three important aspects of Aglets and Concordia performance: mobile agent dispatch from agent servers, mobile agent dispatch from applets, and messaging.

In particular, results from Tests 1 and 5 show that Concordia performs substantially better than Aglets in agent dispatching from agent servers (see the

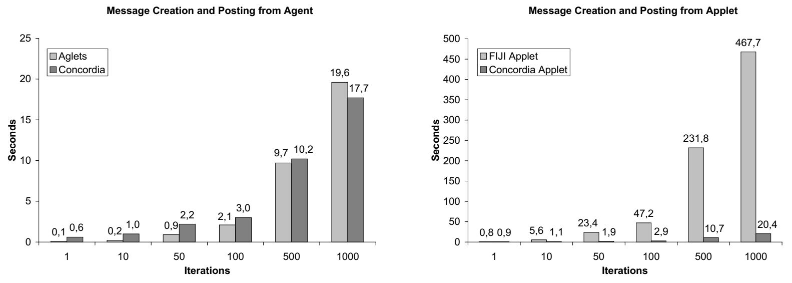


Fig. 4. [MSG] Benchmark; results from Tests 3 and 4.

diagrams on the left side of Figures 3 and 5). This is attributed to the fact that, when transporting an agent, Concordia dispatches an image of it. This image retracts its classes from the sender, on a need-to-use basis. In contrast, Aglets Workbench dispatches an Aglet together with all the objects reachable from this Aglet. Furthermore, it is plausible that additional overhead is incurred by Aglets due to the extra level of indirection introduced by their callback-based transport model.

Concordia performs worse than Aglets, however, when dispatching agents from an applet. This can be attested from Test 2 (Figure 3, right) and from a comparison between the two diagrams of Figure 5; note that in Test 6 the performance difference between Concordia and Aglets is smaller than in Test 5. This is probably due to the fact that the current handling of applets by the Aglets Workbench (through IBM's Fiji Applet and plug-in) is more optimized than the work-around we did to launch and receive Concordia agents from applets. This workaround requires manual installation of Concordia/application classes on the client-machine's local disk. It should be noted that, across both platforms, dispatching agents from applets performs substantially worse than dispatching agents from agent servers.

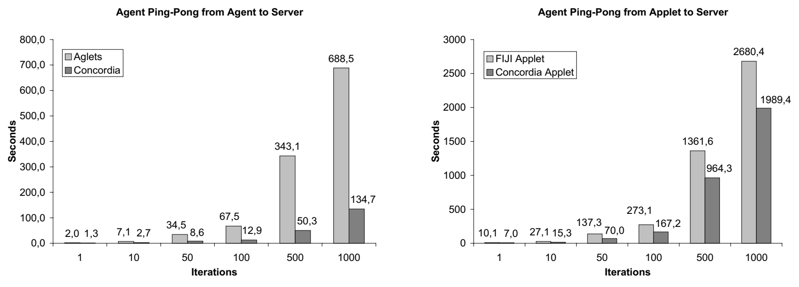


Fig. 5. [ROAM] Benchmark; results from Tests 5 and 6.

Results from Test 3 (Figure 4) show that Aglets outperform Concordia in message creation and posting. Concordia, however, performs better in the case of 1000 iterations. We believe this is an artifact of Tahiti-server performance behavior, which handles the transmission of incoming and outgoing messages. Further tests have shown that Tahiti saturates under heavy load. Concordia, on the other hand, separates the handling of messages (events) from the handling of agents and, apparently, Concordia's Event Manager is better optimized to sustain higher messaging loads than Tahiti. Turning to Test 7 (Figure 6), we observe a significant degradation of Aglets performance under the [SYNC] benchmark, with an increasing number of benchmark iterations. We believe this degradation is related to the implementation of message reply under Aglets, and with the erratic performance of the Tahiti server under heavier loads.

Interestingly, Concordia outperforms Aglets in message transmission from an applet, and in message exchange between an applet and an agent (see Tests 4 and 8, Figures 4 and 6). In our experiments, the use of applets under the Aglets platform has an additional overhead factor, which is due to security limitations: for a FijiApplet to communicate with an agent server (Tahiti), it has to make an extra hop and go through the Web server, where from the applet was downloaded to the client machine. This is not the case with the Concordia work-around we employed to dispatch and receive agents from an applet.

Another interesting observation from micro-benchmark measurements is that, in contrast to Aglets, Concordia performance scales impressively well as we increase the number of benchmark iterations. This is attributed to the way Concordia handles agent transportation by creating and maintaining a persistent image of an agent before dispatching it to another machine. Hence, Concordia avoids class loading on subsequent transfers, whereas Aglets must continuously load the needed classes on every Aglet transfer.

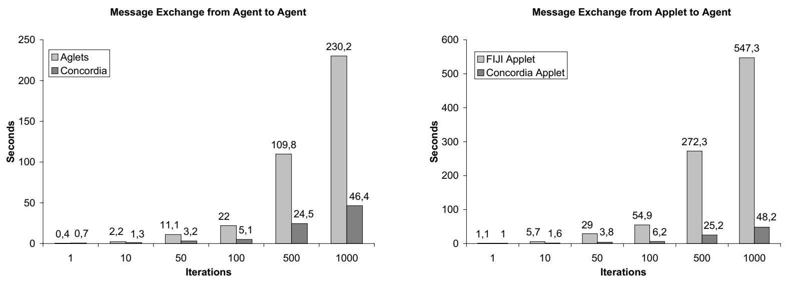


Fig. 6. [SYNC] Benchmark; results from Tests 7 and 8.

5.3 Web Database Micro-Kernels

In this chapter, we focus on micro-kernels that use mobile agents to provide database access over the Web and correspond to the computational models pre-

Table 1. Micro-kernels materialized with Mobile Agents.

Computing Model	Kernel	Comments
C/S-MA	**Framework 1**	*Baseline*
C/A/S-MA	**Framework 2**	*Using messenger agents*
C/A/S	**Framework 3**	*Using messages*
C/A/S-MA-CSB	**Framework 4**	*Using Service Bridges and messenger agents*
C/A/S-CSB	**Framework 5**	*Using Service Bridges and messages*

sented earlier. We propose a *micro-kernel* consisting of a short transaction (three queries) between a client and a remote database. The queries select all entries of a small student database. We measure the time required to launch an agent (or a message) from the client site, the time to carry this agent to the database server, the time to connect to the database and execute the query, and the time to bring the results back to the client. We measure the time to query the remote database for the first time and for any subsequent request. We expect these two measurements to be different, as the time of the first query includes the connection to the database and the downloading of the JDBC drivers from the client or its surrogates.

The kernel is implemented with mobile agents following the computing models **C/A/S** and **C/A/S-MA**. We also implement the kernel according to the "mobile" client-server model (**C/S-MA**). Concordia Service Bridges represent an alternative non-dynamic way to materialize the server-side of the C/A/S and C/A/S-MA models. Therefore, we implement our Web-database kernel with Service Bridges, adding another two frameworks in our benchmark suite. Table 1 summarizes the application kernels employed in our tests and the notation we use for them subsequently.

Tests: We ran tests to evaluate the five frameworks presented in Table 1. Details about the tests are given below. The measurements are presented in Figure 7.

Framework 1 implements the C/S-MA model for Web-database connectivity. The client-side is implemented as an applet, which is downloaded by a client machine with a Java-enabled Web browser. Communication between the client and the server is done through the DBMS-agent launched by the applet. Upon arrival to the server, the DBMS-agent downloads the appropriate JDBC driver and connects to the database. Subsequently, it carries client queries and query results between the client and the remote database.

Framework 2 implements the C/A/S-MA model. The client-side is again implemented as an applet, which is downloaded to a client machine with a Java-enabled Web browser. The applet launches two agents to the database server. One of these agents "parks" to the server and is responsible for downloading the necessary JDBC driver, connecting to the database and querying it. The other agent is the messenger that undertakes the responsibility of transferring the results to the client and the new client requests to the "parked" agent. The "parked" agent is transported and connected to the database server only for the first query.

Framework 3 implements the C/A/S model. Implementation is similar to Framework 2 except for the communication between the client and the database server, which employs messages instead of agents.

Framework 4 implements the C/A/S-MA-CSB model: we created a Concordia Service Bridge that performs the Web access on behalf of an incoming agent. The incoming agent carries the SQL statement from the applet client to the Service Bridge, and returns the results back to the client.

Framework 5: This framework uses events for the communication between the applet and the Service Bridge. Both the applet and the Service Bridge are connected to the Event Manager of the Concordia Server at the database machine and they exchange Access Request events and Access Results events.

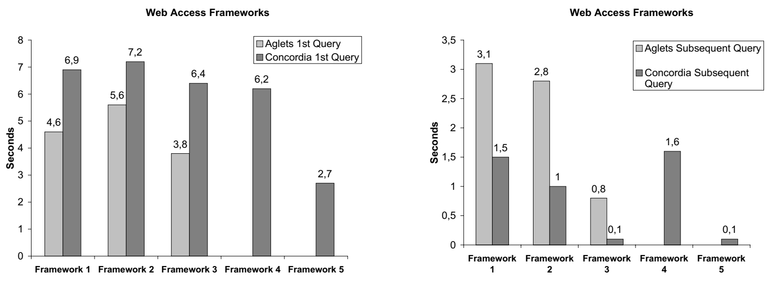


Fig. 7. Performance of Web-database kernel.

Discussion: Figure 7 presents our measurements from the frameworks presented above. A first remark is that the Aglets Workbench outperforms Concordia for the first query, as shown in the left diagram of Figure 7. This observation is consistent with the results of Test 2, and with our remarks on applet-performance under Concordia (Figure 3, right diagram). Concordia, however, outperforms Aglets in subsequent queries. For Frameworks 1 and 2, this observation is consistent with our results from Test 6 (see Figure 5, right diagram). In the case of Framework 3, the improvement of Concordia's performance over Aglets agrees with our [SYNC] micro-benchmark (see Figure 6).

The better performance displayed by Framework 1 with respect to Framework 2 for the first query, is due to the extra overhead incurred by the dispatch of a second agent under Framework 2. This agent parks at the server and results in the improved performance displayed by subsequent queries under Framework 2 over Framework 1.

It is also interesting to point out that the small performance difference between Frameworks 1 and 3 for the first query, is due to the more limited functionality of the agent sent to park at the server-side under Framework 3 (C/A/S model). This agent is "lighter" than the agent dispatched under Framework 1

(C/S-MA model), as it does not have to cope with dispatching itself back and forth to the client. Another interesting point is the performance benefit of using messages over messenger agents. This is expected, though, as messenger agents are "heavier" objects than messages, both in Aglets and Concordia. Messenger agents, however, offer greater flexibility as they can roam the network collecting more information before reaching their destination. Finally, Concordia's Service Bridges represent a very efficient approach for providing services to incoming agents at the server side. This approach, however, lacks the flexibility of dynamically "parking" a mobile agent at the server-side, at run-time, and having this agent negotiate with the server the services it will provide to incoming agents.

6 Conclusions

In this chapter we presented a short overview of various approaches for performance evaluation and analysis of mobile-agent applications and systems. Then, we described a quantitative performance analysis methodology that we developed to investigate performance properties of MA platforms and systems. We implemented this methodology with a hierarchy of benchmarks (micro-benchmarks and micro-kernels), which we ran on top of two popular Java-based mobile-agent platforms, IBM's Aglets and Mitsubishi's Concordia Results from micro-benchmark tests revealed interesting aspects of Aglets and Concordia performance, and enabled us to interpret the performance of micro-kernels. On the other hand, micro-kernel-performance measurements helped us assess the distributed computing models examined. Furthermore, these measurements confirm the validity and usefulness of the proposed micro-benchmarks. As expected, both platforms have their pros and cons, with Concordia providing better performance and robustness and Aglets offering improved flexibility. We are currently developing further benchmarks to extend our approach for assessing the performance of mobile-agent-based distributed applications.

References

1. Internet Performance Modeling. Workshop on Internet Performance Modeling. Department of Computer Science, University of Dortmund, October 1999.
2. Y. Aridov and D. Lange. Agent Design Patterns: Elements of Agent Application Design. In *Proceedings of Autonomous Agents 1998*, pages 108–115. ACM, 1998.
3. N. Bhatti, A. Bouch, and A. Kuchinsky. Integrating user-perceived quality into Web server design. In *Proceedings of the 9th International World-Wide Web Conference*, pages 1–16. Elsevier, May 2000.
4. W. Brenner, R. Zarnekow, and H. Wittig. *Intelligent Software Agents. Foundations and Applications.* Springer, 1998.
5. M. Breugst, I. Busse, S. Covaci, and T. Magedanz. Grasshopper – A Mobile Agent Platform for IN Based Service Environments. In *Proceedings of IEEE IN Workshop 1998*, pages 279–290, Bordeaux, France, May 1998.
6. A. Carzaniga, G.P. Picco, and G. Vigna. Designing Distributed Applications with Mobile Code Paradigms. In *Proceedings of the 1997 International Conference on Software engineering*, pages 22–32, May 1997.

7. A. Castillo, M. Kawaguchi, N. Paciorek, and D. Wong. Concordia as Enabling Technology for Cooperative Information Gathering. In *Japanese Society for Artificial Intelligence Conference*, June 1998. http://www.meitca.com/HSL/Projects/Concordia/.
8. D. Lange and M. Oshima. *Programming and Deploying Java Mobile Agents with Aglets*. Addison Wesley, 1998.
9. M. Dikaiakos and D. Gunopoulos. FIGI: The Architecture of an Internet-based Financial Information Gathering Infrastructure. In *Proceedings of the International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems*, pages 91–94. IEEE-Computer Society, April 1999.
10. M. Dikaiakos, A. Rogers, and K. Steiglitz. Performance Modeling through Functional Algorithm Simulation. In G. Zobrist, K. Bagchi, and K. Trivedi, editors, *Advanced Computer System Design*, chapter 3, pages 43–62. Gordon and Breach, 1998.
11. M. Dikaiakos and G. Samaras. A Performance Analysis Framework for Mobile-Agent Platforms. In *Proceedings of Workshop on Infrastructure for Scalable Mobile Agent Systems, Autonomous Agents 2000*, 2000.
12. M. Dikaiakos and G. Samaras. Quantitative Performance Analysis of Mobile-Agent Systems: A Hierarchical Approach. Technical Report TR-00-2, Department of Computer Science, University of Cyprus, June 2000.
13. M. Dikaiakos and J. Stadel. A Performance Study of Cosmological Simulation on Message-Passing and Shared-Memory Multiprocessors. In *Proceedings of the 10th ACM International Conference on Supercomputing*. ACM, May 1996.
14. J. Dongarra and W. Gentzsch, editors. *Computer Benchmarks*. North Holland, 1993.
15. G. Glass. Overview of Voyager: ObjectSpace's Product Family for State-of-the-Art Distributed Computing. Technical report, ObjectSpace, 1999.
16. J. Gosling and H. McGilton. *The Java Language Environment. A White Paper.* Sun Microsystems. http://java.sun.com/docs/white/index.html.
17. W. Meira Jr., T.J. LeBlanc, and V. Almeida. Using the Cause-Effect Analysis to Understand the Performance of Distributed Programs. In *Proceedings of Symposium on Parallel and Distributed Tools*, pages 101–111. ACM, 1998.
18. N.M. Karnik and A.R. Tripathi. Design Issues in Mobile-Agent Programming Systems. *IEEE Concurrency*, 6(3):52–61, July-September 1998.
19. R. Koblick. Concordia. *Communications of the ACM*, 42(3):96–99, March 1999.
20. David Kotz, Guofei Jiang, Robert Gray, George Cybenko, and Ronald A. Peterson. Performance Analysis of Mobile Agents for Filtering Data Streams on Wireless Networks. In *Proceedings of the Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2000)*, pages 85–94. ACM Press, August 2000.
21. D. B. Lange and M. Oshima. Seven Good Reasons for Mobile Agents. *Communications of the ACM*, 42(3):88–91, March 1999.
22. D.B. Lange and Y. Aridov. *Agent Transfer Protocol – ATP/0.1.* IBM Tokyo Research Laboratory, March 1997. http://www.trl.ibm.co.jp/aglets/.
23. T. Magedanz and A. Karmouch. Mobile Software Agents for Telecommunication Applications. *Computer Communications*, 23(8):705–707, 2000.
24. S. Papastavrou, G. Samaras, and E. Pitoura. Mobile Agents for WWW Distributed Database Access. In *Proceedings of the Fifteenth International Conference on Data Engineering*, pages 228–237. IEEE, March 1999.
25. E. Pitoura and G. Samaras. *Data Management for Mobile Computing.* Kluwer Academic Publishers, 1998.

26. A. Puliafito, S. Riccobene, and M. Scarpa. An Analytical Comparison of the Client-Server, Remote Evaluation and Mobile Agents Paradigms. In *Proceedings of the Joint Symposium ASA/MA '99. First International Symposium on Agent Systems and Applications (ASA '99). Third International Symposium on Mobile Agents (MA '99)*, pages 278–292. IEEE-Computer Society, October 1999.
27. O.F. Rana. Performance Management of Mobile Agent Systems. In *Proceedings of Autonomous Agents 2000*, pages 148–155. ACM, June 2000.
28. O.F. Rana and K. Stout. What is Scalability in Multi-Agent Systems? In *Proceedings of Autonomous Agents 2000*, pages 56–63. ACM, June 2000.
29. G. Samaras, M. Dikaiakos, C. Spyrou, and A. Liverdos. Mobile Agent Platforms for Web-Databases: A Qualitative and Quantitative Assessment. In *Proceedings of the Joint Symposium ASA/MA '99. First International Symposium on Agent Systems and Applications (ASA '99). Third International Symposium on Mobile Agents (MA '99)*, pages 50–64. IEEE-Computer Society, October 1999.
30. L.M. Silva, G. Soares, P. Martins, V. Batista, and L. Santos. Comparing the Performance of Mobile Agent Systems: a Study of Benchmarking . *Computer Communications*, 23(8):769–778, 2000.
31. T. Spalink, J. Hartman, and G. Gibson. The Effects of a Mobile agent on File Service. In *Proceedings of the Joint Symposium ASA/MA '99. First International Symposium on Agent Systems and Applications (ASA '99). Third International Symposium on Mobile Agents (MA '99)*, pages 42–49. IEEE-Computer Society, October 1999.
32. C. Spyrou, G. Samaras, E. Pitoura, and P. Evripidou. Wireless Computational Models: Mobile Agents to the Rescue. In *2nd International Workshop on Mobility in Databases & Distributed Systems. DEXA '99*, September 1999.
33. M. Strasser, J. Baumann, and F. Hohl. Mole - A Java Based Mobile Agent System. In J. Baumann, editor, *2nd ECOOP Workshop on Mobile Object Systems*, 1996.
34. M. Strasser and M. Schwehm. A Performance Model for Mobile Agent Systems. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 97)*, pages 1132–1140, June 1997.
35. D.L. Tennenhouse, J.M. Smith, W. D. Sincoskie, and G.J. Minden. Itinerant Agents for Mobile Computing. *Journal of IEEE Personal Communications*, 2(5), October 1995.
36. Transaction Processing Performance Council (TPC). *TPC Benchmark W (Web Commerce) - Draft Specification*, December 1999.
37. J. White. Telescript Technology: Mobile Agents. In J. Bradshaw, editor, *Software Agents*. MIT Press, 1997.
38. J.E. White. *General Magic White Paper*. http://www.genmagic.com/agents, 1996.
39. D. Wong, N. Paciorek, and D. Moore. Java-based Mobile Agents. *Communications of the ACM*, 42(3):92–95, March 1999.
40. D. Wong, N. Paciorek, T. Walsh, J. DiCelie, M. Young, and B. Peet. Concordia: An Infrastructure for Collaborating Mobile Agents. *Lecture Notes in Computer Science*, 1219, 1997. http://www.meitca.com/HSL/Projects/Concordia/.
41. M. Woodside. Software Performance Evaluation by Models. In C. Lindemann G. Haring and M. Reiser, editors, *Performance Evaluation: Origins and Directions*, pages 283–304. Springer, 1999.