Mobile Agent Platforms for Web Databases: A Qualitative and Quantitative Assessment

George Samaras Marios D. Dikaiakos Constantinos Spyrou Andreas Liverdos
Department of Computer Science
University of Cyprus
P.O. Box 20537, 1678 Nicosia, Cyprus
{cssamara,mdd}@cs.ucy.ac.cy

Abstract

In this paper we present practical experiences gathered from the employment of two popular Java-based mobile-agent platforms, IBM's Aglets and Mitsubishi's Concordia. We present some basic distributed computing models and describe their adaptation to the mobile-agent paradigm. Upon these models we develop a set of frameworks for distributed database access over the World-Wide Web, using IBM's Aglets and Mitsubishi's Concordia platforms. We compare the two platforms both quantitatively and qualitatively. For the quantitative comparison, we propose, employ, and validate an approach to evaluate and analyze mobile-agent framework performance. For the qualitative assessment, we present our observations about the programmability and robustness of, and mobility provided by, the two platforms.

1 Introduction

To better understand the performance behavior of computer systems, it is helpful to define relatively simple *metrics* highlighting particular aspects of performance or particular bottlenecks. Also, to design simple, portable and scalable *benchmarks* that measure how performance metrics vary with respect to *system parameters* and/or *application characteristics*. Numerous studies have assembled and used benchmarks to evaluate the performance properties of existing systems, to enable performance prediction of future architectures, and to model the performance of interesting classes of applications [7, 6, 3]. Notably, performance benchmarking has proven to be complicated in the case of distributed systems, which demonstrate a significant lack in methods and tools for performance analysis.

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 [15, 28, 12]. Mobile agents are being used already in a variety of Internet-based distributed computing applications: Web databases [19], cooperative environments [2], information-gathering systems [5], electronic commerce systems [30], and so on. In that context, a distributed application can be thought of as a dynamic group of agents working in coordination to accomplish some goal.

As the mobile-agent paradigm becomes prevalent in distributed systems, we expect that the quest for programming abstraction, code reuse, portability and performance will lead to the emergence of mobile agent *application frameworks*. A framework will offer a set of basic services (database queries, http requests, Web queries), implementing some distributed computing model. An application developer will choose a computational model according to performance and functionality requirements of the application. Consequently, when trying to analyze the performance of mobile-agent-based distributed systems, we need to employ metrics and benchmarks that take into account: a) the relevant application frameworks and their basic services; b) the underlying computational model, and c) the context of use of the application frameworks and the workload characteristics arising in it. Nevertheless, the identification of metrics and benchmarks for mobile-agent platforms is hard, given the dynamic nature of mobile-agent applications, the applicability of mobile agents in a wide spectrum of resources (LAN, WAN, wireless networks, PCs, workstations) and the fact that mobile agent applications have not reached

yet a level of widespread adoption that will determine which frameworks are successful and popular.

In this paper, we address issues pertinent to the performance analysis of mobile agent-based distributed systems. In particular, we focus on mobile agent frameworks providing distributed database access over the World Wide Web and present several computational models that are pertinent to this application. The choice of Web databases is not random: previous research has established the advantages of the mobile agent approach over other state-of-the-art techniques [19]. We propose a number of benchmarks to assess basic performance properties of mobile agent platforms and compare the performance of different mobile agent-based frameworks for distributed database access over the Web. Finally, we describe our experiences with two Mobile Agent platforms that we used to provide distributed database access over the Web: IBM's Aglets and Mitsubishi's Concordia. We compare the two platforms both qualitatively and quantitatively.

The remaining of the paper is organized as follows: Section 2 presents a brief definition of computational models pertinent to relevant distributed computing applications. Section 3 describes the adaptation of these models to the mobile agent-based, Web-database application. In Section 4 we present two popular Java-based platforms, IBM's Aglets Workbench [4] and Mitsubishi's Concordia [13], and discuss our practical experiences from deploying the two platforms for Web databases. Section 5 presents a performance analysis methodology that we propose to compare the implementation of different frameworks for Web databases. We conclude in Section 6.

2 Mobile-Computing Software Models

The inadequacy of the traditional client-server approach to support wireless and mobile applications has resulted in the development of new computational paradigms. To evaluate emerging paradigms and assess their implementation, we need to identify the computational models upon which these paradigms will be employed and tested. In this section, we describe briefly the various agent-based computational models pertinent to the mobile computing environment. A detailed description of these models, and their strengths and weaknesses, is given in [22].

2.1 Extended Client-Server Models

Extensions of the client-server model are merely based in the introduction of stationary agents placed between the mobile client and the fixed server. The agents alleviate the constraints of the communication link by performing various communication optimizations. Furthermore, the introduction of agents alleviates any client-side resource constraints, by undertaking part of the functionality of resource-poor mobile clients. The degree to which this is achieved depends on the placement and functionality of agents.

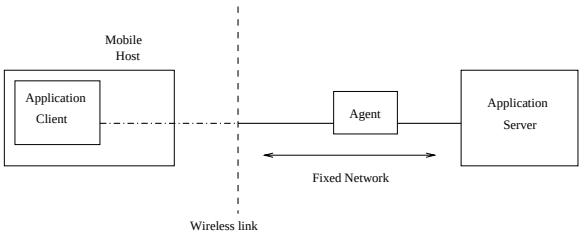

Figure 1. The Client-Agent-Server Model.

The Client-Agent-Server Model: A popular extension to the traditional client-server model is a three-tier model, called client-agent-server (C/A/S) model [26, 17, 8]. The C/A/S model uses a messaging and queuing infrastructure for communication between the mobile client and the agent, and between the agent and the server (see Figure 1). Agents are used in a variety of forms and roles in this architecture. At one extreme, an agent acts as the complete surrogate of a mobile host on the fixed network. In this case, any communication to and from the mobile host goes through the mobile host's agent. At the other extreme, the agent is attached to a specific service or application, e.g., web browsing [10] or database access [17]. Any client's request and server's reply associated with this application is communicated through this service-specific agent. In this scenario, a mobile host must be associated with as many agents as the services it needs access to.

The Client-Intercept-Server Model: The shortcomings of the client-agent-server model are addressed by the deployment of an agent that will run at the mobile device of the end-user, along with the agent of the C/A/S model that runs within the wireline network (see Figure 2) [23, 10] . The client-side agent intercepts client's requests and cooperates with the server-side agent to perform optimizations for reducing data transmission over slow links, to improve data availability and to sustain the non-interruption of the mobile computation.

From the point of view of the client, the client-side agent appears as the local proxy of the server, which is co-resident with the client. Since the pair of agents is virtually inserted in the data path between the client and the server, the model is also called client-

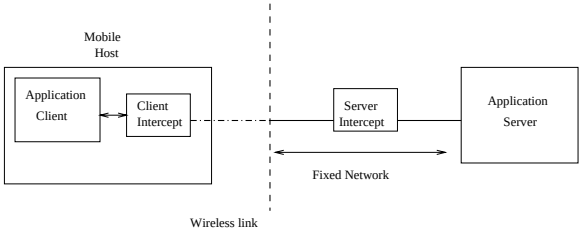

Figure 2. The C/I/S Model.

intercept-server (C/I/S) instead of client-agent-agent-server model [23, 10]. This model is more appropriate for heavy-weight clients with enough computational power and secondary storage to support the client-side agent. The model provides a clear distinction and separation of responsibilities between client and server-side agents. Legacy and existing applications can be executed as before since the agent pair shields them from the limitations of mobility and the wireless media.

2.2 Mobile-Agent Technologies

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 [25, 27, 15, 28]. 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.

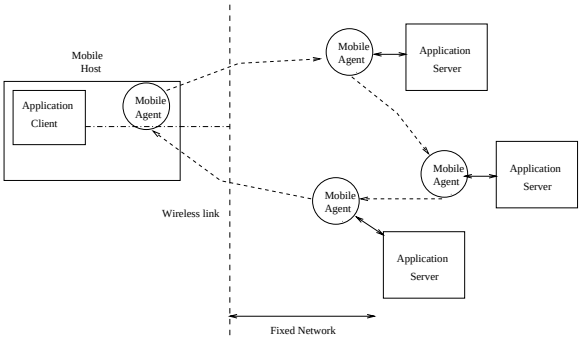

Figure 3. The Mobile Agent Model.

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 [20, 15].

2.3 Mobile Agents and the C/S Model

Mobile agents give rise to new computational models for mobile computing, which we collectively call the *Mobile-Agent Model* (see Figure 3). This model enables a high degree of flexibility as it incorporates the advantages of mobile-agent platforms. It should be noted that the Mobile Agent model is orthogonal to the client-server model and its extensions, since mobile agents can be used to *dynamically materialize* [24] and extend models like the C/S, C/A/S and C/I/S. Such an approach presents many benefits in the wireless and dial-up environments, as well as in the world of Internet services and applications [15].

For example, the server-side agent of the C/A/S and C/I/S model may be seen as a stationary agent, i.e., an agent lacking the ability to migrate to other servers. One can implement these agents, however, as mobile agents that are placed at the client and the server dynamically. Furthermore, the server-side agent may be permitted to move within the fixed network, "following" its associated client, to remain "near" the client and yet within the fixed network. Once the server-side agent starts roaming the fixed network, it can communicate with the client not only via messaging but also via mobile agents. These agents can also roam the fixed network and connect to other servers before returning to the client, thus enhancing the model's flexibility.

The combination of the mobile-agent model with "traditional" software models for mobile computing gives rise to new software models. For instance, the employment of mobile agents for client-server communication, instead of messages, leads to the *"mo-*

bile" client-server (C/S-MA), *"mobile" client-agent-server* (C/A/S-MA) and *"mobile" client-intercept-server* (C/I/S-MA) models.

In these cases we denote as *messenger agents* the mobile agents used by the server and the client to communicate. In such a scenario, the client creates a messenger mobile agent and submits it to the server machine. Upon reception, the server processes the information presented by the messenger agent. In a more flexible approach, a server could refine and extend the messenger agent and then launch it to other servers on the network. When the messenger agent finishes its task, it returns with the results to the server. The server filters out any unnecessary information and transmits to the mobile client only the relevant data. Such an approach entails enhancing servers and clients with capabilities to process mobile agents, i.e. modifying/refining their state and is, in some respect, in accordance with current research on active networks [26]. An extended taxonomy of software models for Mobile Computing, according to the connection modality between clients and servers (client-server, client-agent-server, etc.) and the communication paradigm employed to establish and manage connections, is presented in Table 1.

3 Mobile Agents for Web Databases

The emergence of the World-Wide Web as a universal networking infrastructure for information exchange has created the need and the opportunity for providing distributed-database access over the Web. The current commercial client-server approach to Web databases employs applet-based methodologies for accessing database systems using the JDBC API and JDBC drivers for database connectivity [11]. In most commercial applications, a Web browser downloads an applet from the remote Web server; the applet downloads and initiates a JDBC driver and uses a complex set of JDBC interfaces to connect to the remote SQL server. This approach offers limited flexibility because it is difficult to adapt it to the computational models presented earlier, and it is inadequate in supporting multiple database systems. Furthermore, under the applet-based approach, the client needs to maintain a connection to the server and to download various JDBC classes[1], for the whole duration of the client-server interaction. This reduces its stability and robustness [19]. Within the wireless and dial-up environment these setbacks are further exacerbated.

[1] The size of typical JDBC drivers ranges between 350 and 500 KB.

Improved Web-based distributed access to database systems (e.g., SQL servers) can be established using mobile-agent technologies [19]. In that case, the resulting application frameworks can be formulated in concordance with the computational models described in Section 2, with mobile agents being used to dynamically materialize components of the models.

3.1 C/S-MA for Web Databases

The deployment of mobile-agent models offers an approach to database access over the Web, which is totally different than the applet-based approach. The mobile-agent approach (named C/S-MA) is based on using mobile agents between the client interface and the server machine to provide database connectivity, processing and communication. In particular, the DBMS-applet creates and launches a mobile agent (or agents if necessary) that travels directly to the remote SQL server. At the SQL server, the mobile agent initiates a local JDBC driver, connects to the database and performs any queries specified by the sending client [19]. When the mobile agent completes its task at the SQL server, it dispatches itself back to the client machine directly into the DBMS-applet from where it was initially created and fired. Note that database capabilities are dynamically acquired not at the client but at the server side. Mobile agents that acquire such capabilities are called DBMS-agents [19].

Recent work has explored the performance advantages of using mobile agents for Web-database access versus the traditional approach [19]. By using a DBMS mobile agent (namely the DBMS-Agent) to encapsulate all interactions between the client applet and the SQL server machine, the client applet becomes light and portable. This result is achieved by avoiding the downloading and initialization of JDBC drivers at the client's DBMS-applet. Instead, the DBMS-agent loads the JDBC driver at the SQL server. The single responsibility of the client is to specify the URL address of the database server, the query to be performed, security certificates and an itinerary. The rest is the responsibility of the DBMS-agent. The effect on performance is significant; experiments conducted with IBM's Aglets Workbench [4] have shown that in the wireless and dial-up environments and for average sized transactions, the mobile agents framework improve performance by a factor of two [19]. Even on a fixed network, the employment of the mobile agent model resulted in comparable to the applet-based approach performance.

Notably, the DBMS-agent is independent of the various JDBC driver implementations. The DBMS mobile agent cannot (and is not supposed to) be aware of

Communication Paradigm	Interaction Modality between Clients and Servers			
Messaging	C/S	C/A/S	C/I/S	Mobile
Messenger Agents	C/S-MA	C/A/S-MA	C/I/S-MA	Agent Model

Table 1. Software Models for Mobile Computing.

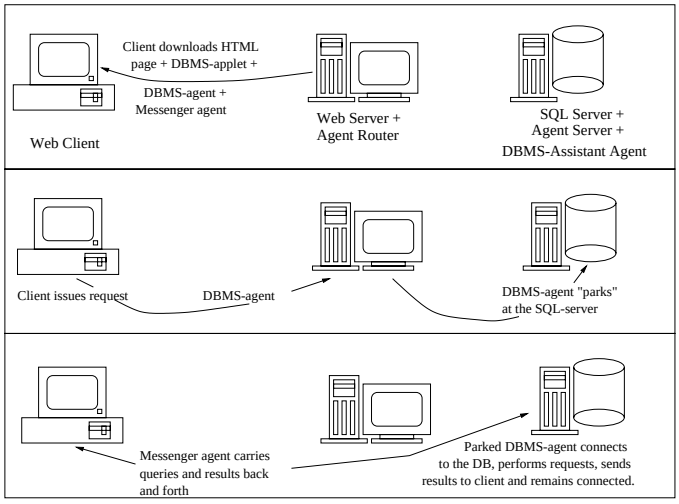

Figure 4. C/A/S-MA for Web databases.

which JDBC driver to load when it arrives at an SQL server. Upon arrival at the SQL server's context, the DBMS-agent is informed of all available JDBC drivers and corresponding data sources. The DBMS-agent is then capable of attaching itself to one or more of these vendor data sources. Finally, besides database connectivity, the functionality of the DBMS-agent may include many other optimizations and tasks, such as query coordination to support the execution of multiple client queries [18], or view materialization to compose a view with information from multiple URLs that are of interest to the client.

In summary, the role of the DBMS-agent in the mobile-agent framework is to convey the various database requests to the SQL server and bring back the result. The proposed mobile-agent-based framework is quite flexible and scalable. It also allows the clients to be lighter, autonomous, and robust. Unfortunately, an agent is fired to the database server and a local JDBC connection is established every time a request is issued, thus introducing an unnecessary and undesirable overhead.

3.2 C/A/S-MA for Web databases

Turning to the C/A/S-MA model, the (server-side) agent of the model can be a mobile agent dynamically created at the client, and then sent and parked at the SQL server. Database connectivity is now the responsibility of the "parked" agent. A JDBC connection can be established and maintained for the whole duration of the client's application, thus eliminating the previous limitation of creating an agent and a connection per request. Between this parked agent and the remote client, another agent carries requests and results back and forth.

Based on this variation (see Figure 4), upon the first client request, two DBMS-agents are fired from the DBMS-applet at the client. The first one is called the "parked DBMS-agent" and is the client's surrogate for database access located on the fixed network. Its role is to "camp" at the SQL server's agent context, load the appropriate JDBC driver, connect to the database, submit requests, and collect and filter the answers. The second agent is the messenger agent. The messenger agent is responsible for carrying the requests and results back and forth to the DBMS-applet. All requests are transmitted to the parked DBMS-agent via the messenger agent. Effectively, the two agents have dynamically materialized the C/A/S-MA model. Furthermore, these two agents may have the same itinerary and thus, if the parked DBMS-agent moves to another server the messenger agent can deterministically follow it, thus dynamically maintaining the client-agent-server model. An additional benefit of this approach is the ability of the messenger agent to roam around the network before returning to the client. Hence, the messenger agent can connect to and interact with other servers before returning to the client.

The effect on performance is quite significant; experiments conducted with IBM's Aglets Workbench [4] have shown that in the wireless and dial-up environments and for average-size transactions, this mobile framework improved performance over the "traditional" applet-based approach by a factor of two to three [19]. It is worth noting that in these environments, C/A/S-MA performed better than the C/S-MA model [19]. Even on a fixed network, the employment of this model resulted in performance comparable to

the applet-based approach.

3.3 C/A/S for Web databases

The traditional C/A/S model can be materialized by letting the client communicate with the agent through messages instead of mobile agents. This is approach achieved by replacing the messenger agent with two types of messages. The first type, delivered from the DBMS-applet to the DBMS-agent, contains the client query and any additional directions to the parked DBMS-agent. The second type, delivered from the parked DBMS-agent to the DBMS-applet, contains the results of the last query. This methodology demonstrates a true service-specific client-agent-server application. The agent is literally inserted into the path between the client and the server communicating with each other via messages. By using a DBMS-agent parked at the server, we avoid the reconnection cost; by using messages instead of the messenger agent we eliminate the time of negotiation and the amount of data transmitted between the client and the server.

The effect on performance is quite significant; experiments conducted with IBM's Aglets Workbench [4] have shown that in the wireless and dial-up environments, and for average-size transactions, the client-agent-server-based framework improved performance by a factor of ten over the "traditional" applet-based approach. Over the C/A/S-MA approach, performance is improved by a factor of three [19]. Even on a fixed network, the employment of this model gave performance gains over the applet-based approach ranging from 30% to 40% [19].

3.4 C/I/S-MA for Web databases

Employing the client-intercept-server model introduces a database-specific agent residing at the mobile client, in addition to the database agent at the fixed network. The "database" agent at the client is called *client-side agent* whereas the database agent at the fixed network is called *server-side DBMS-agent*. While the server-side agent at the fixed network might serve multiple clients, the client-side agent is unique to the client. In contrast to its server-side counterpart, the client-side agent does not need to possess database capabilities (i.e., JDBC connection capabilities). The agent pair cooperates to intercept and control communications over the wireless link for reducing network traffic and query processing.

Functionality at the client-side agent might include various optimizations such as client-side view materialization, caching to support disconnection and weak connectivity, or an asynchronous-disconnected mode, to allow queries that cannot be satisfied by the view to be automatically queued when connectivity is lost and resumed when connectivity is re-established [10, 1].

As in the case of the client-agent-server model, we can implement the client-side agent as a mobile agent. In this case, upon the first client request, the DBMS-applet creates two agents: the client-side agent and the server-side DBMS-agent. The client-side agent remains at the client while the server-side agent is dispatched to the appropriate server. The two agents communicate and cooperate to execute various queries/requests. Both agents are maintained for the duration of the application. Again, communication can be performed either via agents (i.e., the C/I/S-MA model) or via messages (i.e., the C/I/S model).

The C/I/S model(s) offer more flexibility than the C/A/S model(s). For the specific application of Web-database access, however, the resulting performance is identical to that of the C/A/S model(s) because no extra collaboration is needed between the two agents for processing SQL queries.

4 Aglets versus Concordia

4.1 IBM Aglets

The Aglets Software Development Kit (ASDK) is an environment for programming mobile Internet agents (Aglets) in the Java programming language [9]. 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 preinstalled Tahiti server, which is an Aglet server program implemented in Java. A running Tahiti server listens to the host's ports for incoming Aglets and messages. Tahiti captures arriving Aglets, following a First Come First Served policy, 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 transportation procedure is as follows [4]: Calling method `dispatch` on an Aglet will immediately lead to the invocation of its `onDispatching` method. On completion of this method, all threads created by the given Aglet are killed and the Aglet is transferred to its destination. The Tahiti server of the destination captures the Aglet and invokes the `onArrival` method, followed by the `run` method. Meanwhile, the Aglet's new proxy is sent by the destination Tahiti Server to the Tahiti Server of the origin. Communication between the Aglet and the origin Server is conducted throught that proxy.

The protocol used for the transportation of the Aglet, is the Aglet Transfer Protocol (ATP) by IBM, which is an application-level protocol for distributed agent-based systems [14]. ATP defines four standard request methods for agent services, which are `dispatch`, `retract`, `fetch` and `message`. Currently, the Aglets framework does not attempt to optimize the transfer of an agent's bytecode. The transport protocol does not perform checks to find whether it needs to transfer a given class bytecode to a given destination. In every transfer, all the needed classes are transferred. If, however, certain classes are in the destination cache, the Aglets framework utilizes those instead of the ones just transferred, thus improving transformation and downloading time.

A very important and useful property of Aglets is their ability to communicate with each other. Interaglet communication is supported by an object-based messaging framework that is location-independent, extensible and synchronous/asynchronous. This framework is based on a simple callback scheme that requires an Aglet to implement handlers only for the kinds of messages that is supposed to understand. The message callback method in the Aglet class is the `handleMessage`. You don't call this method directly when you wish to send a message to an Aglet. Instead, you invoke either the `sendMessage` for synchronous, or `sendAsyncMessage` method for asynchronous messaging, on the proxy which serves as a message gateway for the Aglet. One of the benefits of using the proxy is that it provides the programmer with a location-independent interface for sending messages to Aglets, because the interface is the same, regardless of whether you are using a remote or a local proxy to send a message. It should be noted that

The Aglet technology does not provide any Managers (until now), so the programmer is forced to implement any managers that is going to use. Managers can be implemented as separate agents.

The Tahiti server (currently version v1.1b, we use v1.0.3 with size 2.25MB) can be installed at any platform that supports Java Virtual Machine (JVM). This means that before installing Tahiti at a machine, you have to successfully install on it the Java Development Kit (JDK) or Java Runtime Environment (JRE).

Fiji Applet is an abstract applet class of a Java package called "Fiji Kit", which allows Aglets to be fired from applets. The FijiApplet maintains an Aglet context from which Aglets can be created, dispatched, and retracted back, but not dispatched to it. For a Java-enabled Web browser (like Netscape Communicator) to host and fire Aglets, two more components are required and are provided by IBM. These are: 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.

4.2 Mitsubishi's Concordia

Concordia is a framework for the development, execution and management of mobile agent applications written in Java [9]. The design goals of Concordia have focused on providing complete coverage of flexible agent mobility, support for agent collaboration, persistence of agent state, reliable agent transmission, agent security, and the ability to add intelligence [13, 29].

The Concordia system is made up of several integrated components. The Concordia server 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 lifecycle of the agent.

Transmission is made possible by the Conduit Server, which is part of the Agent Manager. An agent program initiates its transfer by invoking the local Conduit Server's methods. The agent's execution is suspended and a persistent image of it is created. The Conduit Server inspects the agent's Itinerary Object to determine its destination; then, it proceeds to send an image of the agent to the Conduit Server of the Concordia System at the destination machine. There, the agent is again stored persistently before the acknowledgment of its receipt.

The Queue Manager manages inbound and outbound queues for reliable transport of agents across a network. The Queue Manager communicates with its local Conduit Server and performs handshaking with other remote Queue managers for reliable agent transmission.

Communication in Concordia relies on the Java RMI system [16] which allows an object running in one Java Virtual Machine (JVM) to invoke methods on an object running in another JVM. 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.

To run Concordia agents one has to first download and install the Concordia System (currently version 1.1.2) available as a self-extracting file which includes the Concordia Server Components, the Java

Runtime Environment (JRE), Concordia Documentation and Concordia examples. There are two editions, the free evaluation kit used for our evaluation and the full-blown version. Concordia currently runs on Microsoft's Win32 operating systems and on Solaris.

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. Applet security restrictions do not allow the successful loading and initialization of the applet from a Web server, however, because `AgentTransporter` uses local resources, thus violating security restrictions. To overcome these problems we had to install manually a number of class files at the client machine.

Concordia implements interagent messaging through the concept of events, which are Java objects posted at the Event Manager of a Concordia Server. An object, mobile or stationary, wishing to post or receive particular types of events, must connect to, and register with, the Event Manager of one or more Concordia Servers. The Event Manager keeps track of agent movements and takes care of event distribution. Furthermore, Concordia implements a framework for synchronous or asynchronous agent collaboration. Last but not least, it provides a powerful way of expanding the potential of agents by allowing them to interact with Java applications installed on a Concordia node. These applications are called *Service Bridges* and act as gateways between Concordia and 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. With this directory running, agents need not explicitly know where each Service Bridge resides as they can make their requests through the directory.

4.3 A qualitative comparison

Programming using Aglets Software Development Kit (code writing and debugging) is simple and easy because of Aglet's class structure. The programmer has to override specific methods (e.g., `onCreation`, `onDispatching`, `onArrival`, etc.) to specify agent behavior. We had some problems deploying applications, but these were solved easily thanks to the examples, documentation, and the lively Aglet's mailing list (aglets@kdel.info.eng.osaka-cu.ac.jp).

Although our Aglet applications were generally quite robust, we often experienced Aglet router failures and, less often, Tahiti server failures. We believe that this problem is due to implementation specifics and machine configuration. We also observed that Aglet systems are more robust running under the UNIX operating system.

Developing Concordia applications (code writing and debugging) was also quite easy. The examples and documentation shipped with Concordia were helpful. We had problems deploying Concordia applications because of insufficient documentation regarding the known problems and their work-around, such as the `ConcordiaApplet` security problem, the fact that an applet cannot send an agent directly to a Concordia Server but only to an `AgentTransporter`. There is a mailing list supporting the Concordia community, but it is not very active. Our Concordia-based applications were robust.

Both platforms support weak mobility in the sense that, during an agent transfer, only the agent state is serialized and not the execution stacks and program counters of the various threads executed by the agent at transfer time. Consequently, all threads are terminated before a transfer occurs. This, is due to JVM's built-in security model which does not allow a program (and for that matter any Java-based mobile agent, a Concordia agent or Aglet) to directly access and manipulate the execution stack. Thus, while in theory an agent should be able to migrate with all its state (i.e., heap, execution stack, and registers), in reality Concordia and Aglets are only able to serialize/trasfer only the data on the heap (i.e., its instance variables), essentially following a weaker notion of mobility.

Concordia is based on RMI and provides a more optimized agent transfer protocol than Aglet's ATP. Concordia transfers only an image of the agent; all other objects are transfered on a need-to-use basis. In contrast, Aglets Workbench transfers all the objects reacheable by the agent object, in each and every transportation of this agent. Note that, in Concordia, once the necessary classes are transferred to the destination, no other transfer is required since subsequent agent transportation finds those classes in the local cache.

Aglets provide a more structural programming model that is based on the "callback" methodology: before any major event in an Aglet's life, a "callback" method is invoked to allow the Aglet to prepare for that event. For example, before an Aglet is actually dispatched to a new location, the Aglet's `ondispatch` method is invoked. This method indicates to the Aglet that is about to be send to a new host. In the body of `ondispatch`, the Aglet code must decide whether to go or not. If it decides to go, it must complete any

unfinished business and prepare its state for serialization/transfer. While this methodology provides some benefits, this level of indirection handicaps its performance. Concordia, on the other hand, directly specifies the method to be executed as a result of a major event (dispatch, etc.). This increases performance but minimizes the agent's flexibility in preparing itself for a major event.

An important issue when comparing mobile-agent platforms is their compatibility with Java applets. Applets offer the flexibility of running on any machine with minimum configuration (a Web browser and a plug-in). Concordia applets proved more powerful than Fiji Applets as they can receive any agent launched from anywhere and send and receive events in the same manner. Fiji applets can only retract Aglets or communicate with Aglets that they launched. Of course, Fiji applets are more secure and flexible, as they can download the application classes dynamically from the Web server; this is not possible with the current version of Concordia, which requires the application classes to be preloaded on the client.

5 A Performance Analysis Framework

Performance evaluation of distributed systems and applications is hard for several reasons:

- The absence of global time, control, and state information in distributed systems, which makes performance measurements hard.
- The heterogeneity and complexity of distributed platforms, which makes it difficult to characterize key system performance aspects via small sets of metrics.
- The existence of a great variety of computing models adopted by different distributed applications: client server, client-agent-server, mobile agent, etc. Notably, a change in the computing model may require the adoption of different performance metrics and representative benchmarks.
- The diversity of operations found in real distributed applications, which makes it hard to model typical applications through simple and portable benchmarks [21].
- The "fluidity" of distributed computing systems configuration, especially in the presence of mobility. This fluidity prevents the representation of available system resources with small and simple sets of parameters.

These factors are further exacerbated when deploying Internet-based distributed systems built on top of Java-based mobile-agent platforms, because of the extra issues affecting performance, like the JVM, garbage collection, etc. To study the performance of this kind of distributed system we need to adopt appropriate **metrics** representing the performance of key components of systems and applications. Also, to provide **benchmarks** that will: a) enable the performance characterization of key hardware and/or software components of a distributed system under realistic workload conditions; b) analyze the interplay of system and application parameters with application performance, and c) model the overall performance of important and interesting classes of distributed applications. In this context, benchmarks for mobile agent-based distributed systems can be classified in three categories according to their complexity and utility:

- *Micro-benchmarks*: Short codes designed to isolate and measure basic performance properties of a distributed system, corresponding to key, "low level" system parameters [3]. For example, codes measuring the performance of mobile-agent dispatch, inter-agent communication and synchronization.
- *Application kernels*: Short, synthetic codes designed to measure the basic performance properties of application frameworks of interest; e.g., codes that will assess the performance of Web database access through a particular client-server model. The conjecture here is that the performance of these frameworks will determine to a large extent the overall performance of distributed applications built on top of them. Thus, application kernels can be used to model overall application performance.
- *Full application codes*: as it happens with parallel system benchmarks, kernel performance is only indicative of the overall application performance [3]. Consequently, it is often useful to analyze the performance of distributed systems executing real applications with realistic workloads.

At this stage, it is premature to devise a set of generic benchmarks covering most aspects of mobile-agent performance, as mobile-agent platforms have not being tested widely in real-world applications. Hence, to compare the performance of different mobile-agent platforms, we explore a set of microbenchmarks. Furthermore, we employ simple application kernels to assess the relative performance of application frameworks providing access to Web-based databases deployed on Internet.

5.1 Microbenchmarks

As described in Sections 2 and 3, computing models that use mobile agent 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, microbenchmarks 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 [7]).

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.

Microbenchmark Experiments: For the quantitative comparison we ran several tests implementing the microbenchmarks 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.

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 microbenchmark 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.

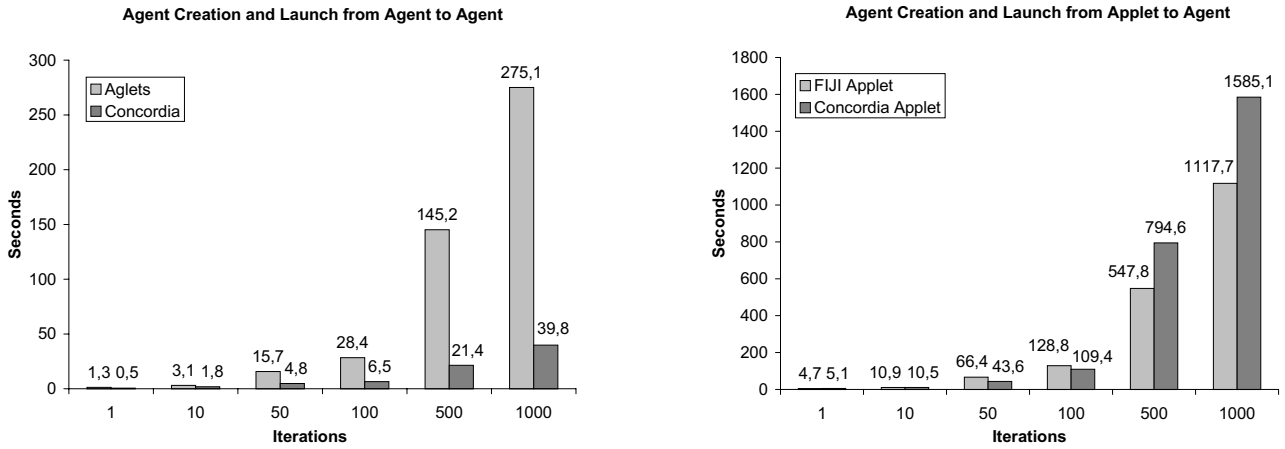

Figure 5. [AC/L] Benchmark; results from Tests 1 and 2.

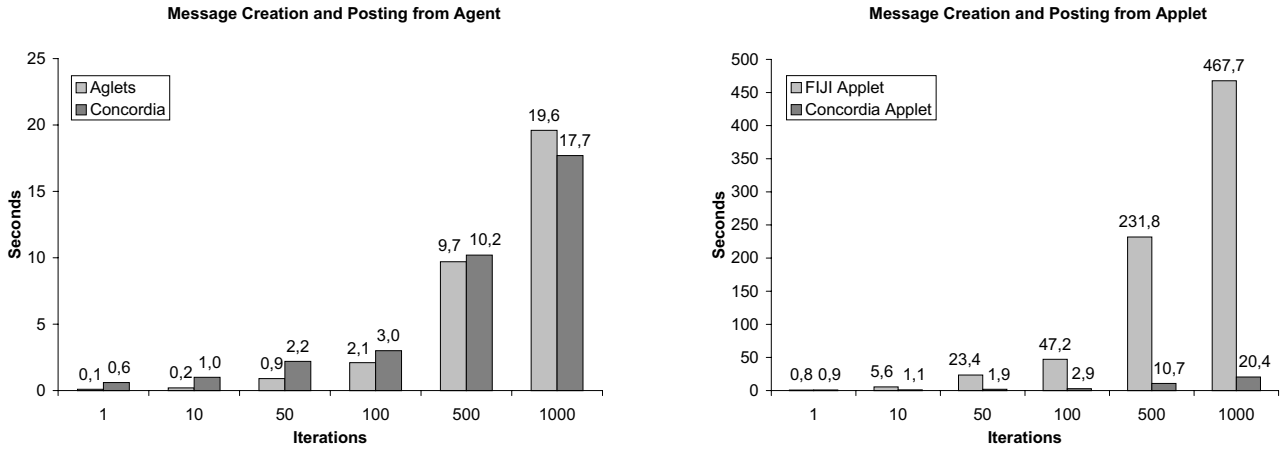

Figure 6. [MSG] Benchmark; results from Tests 3 and 4.

In particular, results from Tests 1 and 5 show that Concordia performs substantially better than Aglets in agent dispatching from agent servers (see the diagrams on the left side of Figures 5 and 7). 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 reacheable 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 model.

Concordia performs worse than Aglets, however, when dispatching agents from an applet. This can be attested from Test 2 (Figure 5, right) and from a comparison between the two diagrams of Figure 7; 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 workaround 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.

Results from Test 3 (Figure 6) 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

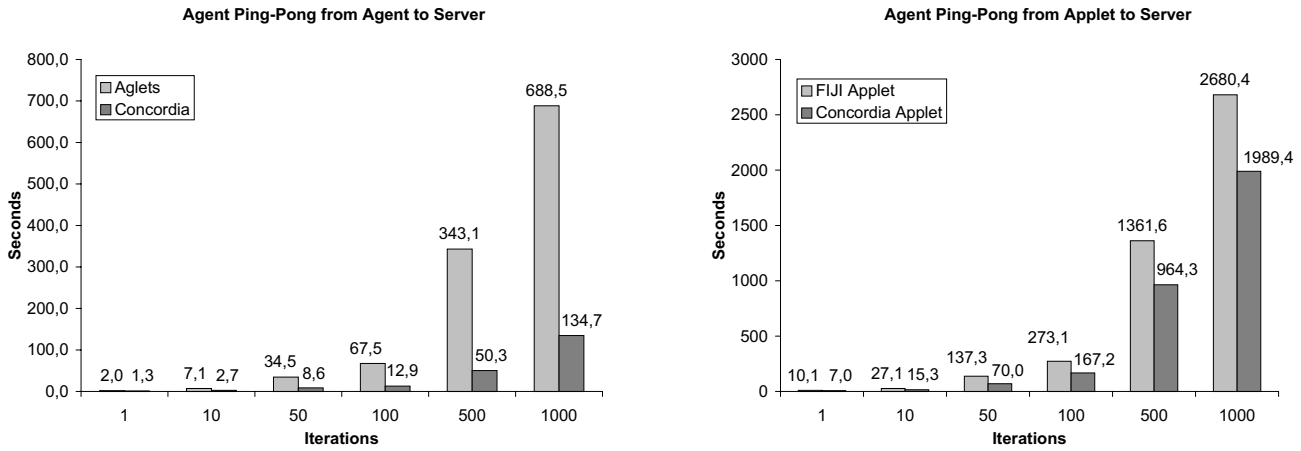

Figure 7. [ROAM] Benchmark; results from Tests 5 and 6.

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 8), 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. We are currently investigating this hypothesis.

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 6 and 8). 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 microbenchmark 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. We are currently investigating this further.

5.2 Web database Application Kernels

In this paper we focus on application frameworks that use mobile agents to provide database access over the Web and correspond to the computational models presented earlier. We propose an *application 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 2 summarizes the application frameworks employed in our tests and the notation we use for them subsequently.

Tests: We ran tests to evaluate the five frameworks presented in Table 2. Details about the tests are given

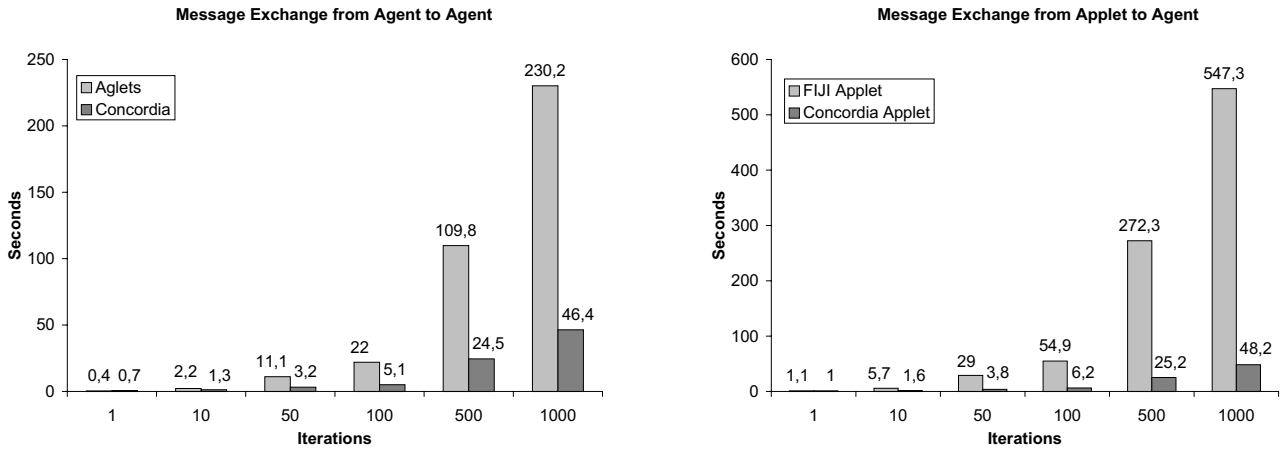

Figure 8. [SYNC] Benchmark; results from Tests 7 and 8.

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*

Table 2. Application Kernel materialized with Mobile Agents.

below. The measurements are presented in Figure 9.

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.

Discussion: Figure 9 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 9. This observation is consistent with the results of Test 2, and with our remarks on applet-performance under Concordia (Figure 5, 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 7, right diagram). In the case of Framework 3, the improvement of Concordia's performance over Aglets agrees with our [SYNC] microbenchmark (see Figure 8).

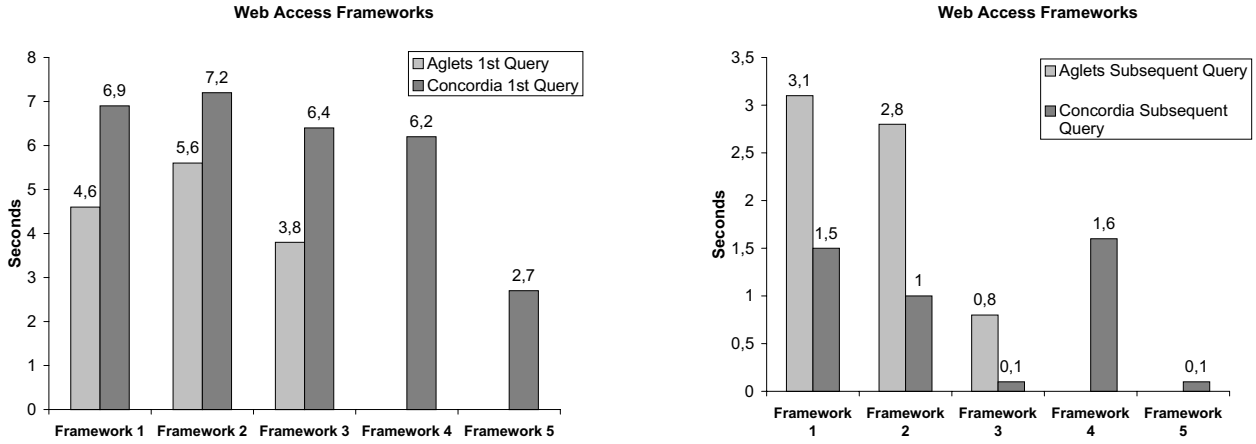

Figure 9. Performance of Web-database kernel.

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 paper we describe practical experiences gathered from the employment of two popular Java-based mobile-agent platforms, IBM's Aglets and Mitsubishi's Concordia, to provide distributed database access over the World-Wide Web. To this end, we formulate some basic distributed computing models and describe their adaptation to the mobile-agent paradigm. Upon these models we develop a set of frameworks for distributed database access over the World-Wide Web, using IBM's Aglets and Mitsubishi's Concordia platforms. We compare the two platforms and the frameworks both quantitatively and qualitatively.

For the quantitative comparison, we propose and apply an approach to evaluate and analyze mobile-agent performance, based on simple microbenchmarks and more elaborate application kernels. Results from microbenchmark tests reveal interesting aspects of Aglets and Concordia performance, and enable us to interpret the performance of application kernels. On the other hand, application-kernel-performance measurements help us assess the distributed computing models examined. Furthermore, these measurements confirm the validity and usefulness of the proposed microbenchmarks. For the qualitative assessment, we present our practical observations from the point of view of programmability, robustness and mobility provided by the two platforms.

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.

Acknowledgments

The authors wish to thank David Kotz for his constructive comments and suggestions on this paper.

References

[1] B. Badrinath, A. Bakre, T. Imielinski, and R. Marantz. Handling Mobile Clients: A Case for Indirect Interaction. In *Proceedings of the 4th Workshop on Workstation Operating Systems*, October 1993.

[2] 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/.

[3] D. Culler, J. Singh, and A. Gupta. *Parallel Computer Architecture. A Hardware/Software Approach.* Morgan Kaufman, 1999.

[4] D. Lange and M. Oshima. *Programming and Deploying Java Mobile Agents with Aglets.* Addison Wesley, 1998.

[5] M. D. 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.* IEEE-Computer Society, April 1999.

[6] M. D. 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.

[7] J. Dongarra and W. Gentzsch, editors. *Computer Benchmarks.* North Holland, 1993.

[8] A. Fox, S. Gribble, E. Brewer, and E. Amir. Adapting to Network and Client Variability via On-Demand Dynamic Distillation. In *Proceedings of the ASPLOS-VII*, 1996.

[9] J. Gosling and H. McGilton. *The Java Language Environment. A White Paper.* Sun Microsystems. http://java.sun.com/docs/white/index.html.

[10] B. Housel, G. Samaras, and D. Lindquist. WebExpress: A Client/Intercept Based System for Optimizing Web Browsing. *ACM/Baltzer Journal of Mobile Neworking and Applications (MONET), Special Issue on Mobile Networking on Internet*, pages 419–431, 1999.

[11] J. Farley, editor. *Java Distributed Computing.* O' Reilly, 1998.

[12] J. Kiniry and D. Zimmerman. A Hands-on Look at Java Mobile Agents. *IEEE Internet Computing*, 1(4):21–30, July-August 1997.

[13] R. Koblick. Concordia. *Communications of the ACM*, 42(3):96–99, March 1999.

[14] D. Lange and Y. Aridor. *Agent Transfer Protocol – ATP/0.1.* IBM Tokyo Research Laboratory, March 1997. http://www.trl.ibm.co.jp/aglets/.

[15] D. B. Lange and M. Oshima. Seven Good Reasons for Mobile Agents. *Communications of the ACM*, 42(3):88–91, March 1999.

[16] S. Microsystems. *Java Remote Method Invocation - Distributed Computing for Java.* Sun Microsystems. http://java.sun.com/docs/white/index.html.

[17] Oracle. *Oracle Mobile Agents Technical Product Summary.* Oracle., June 1997. http://www.oracle.com/products/networking/mobile/agents/html/.

[18] C. Panayiotou, G. Samaras, E. Pitoura, and P. Evripidou. Parallel Computing Using Java Mobile Agents. In *25th Euromicro Conference, Workshop on Network Computing.*, September 1999.

[19] 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.

[20] E. Pitoura and G. Samaras. *Data Management for Mobile Computing.* Kluwer Academic Publishers, 1998.

[21] F. Raab, W. Kohler, and A. Shah. Overview of the TPC Benchmark C: The Order-Entry Benchmark. http://www.tpc.org/cdetail.html.

[22] G. Samaras, E. Pitoura, and P. Evripidou. Software Models for Wireless and Mobile Computing: Survey and Case Study. Technical Report TR-99-5, Department of Computer Science, University of Cyprus, March 1999.

[23] G. Samaras and A. Pitsillides. Client/Intercept: A Computational Model for Wireless Environments. In *Proceedings of the 4th International Conference on Telecommunications (ICT '97)*, pages 1205–1210, April 1997.

[24] 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.

[25] D. Tennenhouse, J. Smith, W. D. Sincoskie, and G. Minden. Itinerant Agents for Mobile Computing. *Journal of IEEE Personal Communications*, 2(5), October 1995.

[26] D. Tennenhouse, J. Smith, W. D. Sincoskie, and G. Minden. A Survey of Active Network Research. *Journal of IEEE Communication Magazine*, 35(1):80–86, January 1996.

[27] J. White. *General Magic White Paper.* http://www.genmagic.com/agents, 1996.

[28] D. Wong, N. Paciorek, and D. Moore. Java-based Mobile Agents. *Communications of the ACM*, 42(3):92–95, March 1999.

[29] 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/.

[30] G. Yamamoto and Y. Nakamura. Architecture and Performance Evaluation of a Massive Multi-Agent System. In *Proceedings of the 3rd Annual Conference on Autonomous Agents*, pages 319–325, May 1999.