The PaCMAn Metacomputer: Parallel Computing with Java Mobile Agents

Paraskevas Evripidou, George Samaras, Christoforos Panayiotou [a]
Evaggelia Pitoura [b]

[a] *Department of Computer Science*
University of Cyprus
P.O. Box 20537 CY-1678 Nicosia, Cyprus
{skevos,cssamara}@ucy.ac.cy

[b] *Department of Computer Science, University of Ioannina*
GR 45110, Ioannina, Greece, pitoura@cs.uoi.gr

Abstract

The PaCMAn (Parallel Computing with Java Mobile Agents) Metacomputer launches multiple Java-mobile Agents that communicate and cooperate to solve problems in parallel. Each mobile agent can travel anywhere in the web to perform its tasks. A number of brokers/load forecasters keep track of the available resources and provide load forecast to the clients. The clients select the servers that they will utilize based on the specific resource requirements and the load forecast. The PaCMAn mobile agents are modular: the mobile shell is separated from the specific task code of the target application. To this end, we introduce the concept of TaskHandlers which are Java objects capable of implementing a particular task of the target application. TaskHandlers are dynamically assigned to PaCMAn's mobile agents. We have developed and tested a prototype system with several applications such as parallel web querying, a prime number generator, the trapezoidal rule and the RC5 cracking application. Our results demonstrate that PaCMAn provides very good parallel efficiency.

1 Introduction

Our motivation is the development of a system that can harness the vast resources of the Internet that would have otherwise be idle into a dynamic heterogeneous Metacomputer for High Performance Computing (HPC). The PaCMAn Metacomputer [11] consists of three major components: servers, clients and brokers. The clients launch multiple Java mobile agents [1,4,5] that travel

Preprint submitted to Elsevier Science *28 February 2000*

in designated servers in the web where they perform their specific tasks. The broker keeps the registry of the servers and clients that participate in the Metacomputer. Each agent supports the basic communication and synchronization tasks of the classical parallel worker assuming the role of a process in a parallel processing application. There is also a second degree of parallelism achieved by the use of multithreading inside each agent that strengthens the concurrent execution of an application. Furthermore, in the PaCMAn framework, we generalize the parallel agent by separating the mobile shell from the specific task code of the target application. We achieve this with the introduction of TaskHandlers, which are Java objects capable of performing the various tasks. TaskHandlers are dynamically assigned to our agents.

The driving force motivating the use of Java mobile agents in PaCMAn is twofold. First, mobile agents provide an efficient, flexible and asynchronous method for obtaining services in rapidly evolving networks. Second, mobile agents support intermittent connectivity, slow networks, and lightweight devices. PaCMAn applications can be started from lightweight devices such as laptop computers or even PDAs. After launching the PaCMAn application, the lightweight device (client) can disconnect from the network and the application will continue executing uninterrupted. The client will receive the output of the application it has launched, whenever it connects to the network again. Thus, the PaCMAn framework is appropriate for the emerging wireless environments.

At any given time, there is a vast number of computers connected to the Internet that are idle. The collective computational power of these computers dwarfs the computational power of all the world's high performance computers put together. The topology of these loosely coupled resources changes very dynamically. These are the resources that PaCMAn is targeting. Traditional parallel processing techniques such as MPI and PVM are better suited for tightly-coupled systems such as multiprocessors and NOWs with high performance networks. A strong point of MPI and PVM is the use of highly optimized conventional languages such as Fortran, C and C++ that have better computational performance than Java. However, this might change in the future. There are several attempts such as the Java Grande Forum [26] to make Java a better environment for HPC than the traditional languages.

PaCMAn also has an advantage over conventional techniques when there are large volumes of distributed data. An ideal application for PaCMAn is distributed data mining. The ability of PaCMAn to launch any number of mobile agents to travel in the web and perform tasks in parallel makes it ideal for application that have huge volumes of distributed data. The 1997 press release of the RSA code breaking challenge stated among others [24]:

> "Perhaps a cure for cancer is lurking on the Internet"

Distributed data mining with PaCMAn on the constantly updated data warehouses of medical institutions worldwide could be utilized to explore such assertions.

Most of the existing web-based Metacomputer projects [15,22,23] are applet based. Thus, the server machines are randomly becoming available when the computer owner points to the web page that hosts these systems. PaCMAn on the other hand is proactive. At any given time, a client can use all the servers that are registered as available at that time. For example, companies can register their computer as available after workhours on weekdays and for the entire weekends. Home computers on the other hand can register as available during work hours. Of course, users have the ability to make their computer unavailable explicitly by turning off the PaCMAn daemon or implicitly when the local load reaches a predetermined threshold.

The current implementation of PaCMAn is based on the Aglet Technology [2], developed by IBM Tokyo, which is a Java-based framework for building mobile agent. We have also tested other mobile agent systems [7] and we are currently porting PaCMAn to the Voyager [25] system which is RMI and CORBA based.

Section 2 presents a brief overview of mobile agents and makes a qualitative assessment of the mobile agents characteristic that make them suitable for PaCMAn. Section 3 presents the basic characteristics of the Aglet Technology. In Section 4, we describe the PaCMAn framework and in Section 5, we focus on the object definitions of the framework. In Section 6, we present our prototype implementation, experimental results from sample applications and performance analysis. Section 7 briefly reviews other approaches that use Java for High Performance Computing. In Section 8, we present our concluding remarks.

2 Mobile Agents

A software agent is an autonomous program that resides on computers and/or networks. At any point it has the ability to interact with its environment, and adapt its course of action in a way that maximizes its progress towards its goal.

Mobile agents are self-contained agents that can navigate autonomously. At each site that they visit, they can perform a variety of tasks [13]. The underlying computational model resembles the multithreading environment in the sense that like individual threads, each agent consists of program code and a state. Communication is usually performed through the exchange of messages. Mobility is a fundamental extension of this model: each agent can autonomously relocate itself or its clones.

To perform useful work, each mobile agent needs to interact with the server environment it visits. Thus, a daemon like interface (i.e. an agent execution environment) is provided at the server site that receives the visiting agents, interprets their state and sends them to other daemons as necessary. This allows the visiting agent to access services and data on the current server. Each daemon must be explicitly started at each site that wishes to participate in the mobile-agent based Metacomputer. Issues of security and authentication can also handled by the agent system interface.

Java mobile agents, the most popular kind of mobile agents nowadays, inherit the universal portability and seamless interface of the Java programming language. Furthermore, their execution environment is the Java Virtual Machine (JVM) that is constantly being improved and optimized. PaCMAn and other similar systems for distributed processing over the web benefit from this increasing investment in Java and the Web in general.

One question that may arise is why utilizing mobile agents and not just mobile objects [6]. The main difference between the two is that agents are autonomous and more flexible entities with dynamic behavior. Have we had used mobile objects, we would had to extend them extensively in order to attain the dynamic behavior required by PaCMAn. For example, the concept of dynamic TaskHandlers could not be supported by mobile objects directly and would have required considerable extensions (which in essence would have amounted in us constructing our own version of rudimentary mobile agents).

The fact that we based our system on mobile agents give us more flexibility to expand in other application areas such as data mining and tasks that combine intelligent behavior and computation. It is in our future plans to introduce more "intelligent" communication in our system, following the KQML/FIPA standard.

3 The Aglet Technology: A Java based Agent Execution Environment

The Aglet Technology [2,3] (also known as the Aglets Workbench) is a framework for programming mobile network agents in Java. Developed by the IBM Japan research group, the Aglet technology was first released in 1996. From a technical point of view, the IBM's mobile agent called **Aglet** (**Ag**ile app**let**) is a lightweight Java object that can move autonomously from one computer server to another for execution, carrying along its program code and state as

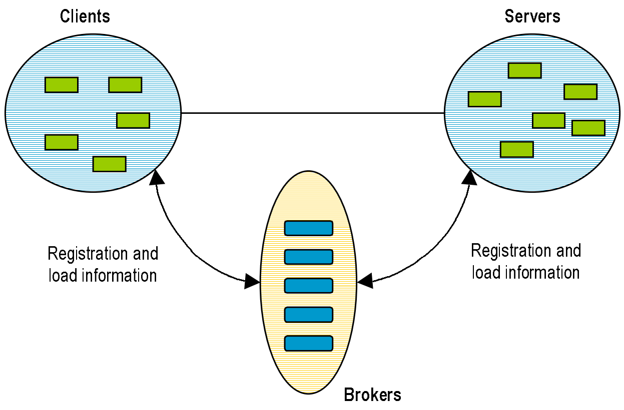


Fig. 1. The PaCMAn Metacomputer

well as any data obtained so far.

An Aglet can be dispatched to any remote server that supports the Java Virtual Machine. This requires that the remote server has preinstalled the Tahiti daemon, a tiny Aglet server program implemented in Java. A running Tahiti server listens to the server's ports for incoming Aglets, captures them, and provides them with an Aglet context (i.e., an agent execution environment) in which they can run their code from the state that it was halted before they were dispatched. Within its context, an Aglet can communicate with other Aglets, collect local information and when convenient halt its execution and be dispatched to another server. An Aglet can also be cloned or disposed.

To allow Aglets to be launched from within applets, the IBM Aglet team provided the so-called FijiApplet, an abstract applet classes that is part of a Java package called Fiji Kit. The FijiApplet maintains some kind of an Aglet context (like the Tahiti Aglet server). From within this context, Aglets can be created, dispatched from and retracted back to the FijiApplet.

For a Java-enabled Web browser to host and launch Aglets to various destinations, besides the FijiApplet applet class, two more components are required and provided by the Aglet Framework. These are an Aglet plug-in to allow the browser to host Aglets and an Aglet router that must be installed at the Web server. The Aglet router is implemented in Java and is available as a stand-alone Java program or as a servlet component for Web servers that support servlets[1]. Its purpose is to capture incoming Aglet and forward them to their destination.

The Aglet framework provides security guarantees in addition to the standard Java security. The security manager of the Tahiti server performs various checks on a visiting agent before it is allowed to run. Furthermore, if during

[1] Servlet: a server-side applet that dynamically extends the functionality of a Web server.

execution, the Aglet requires some protected services (such us disk access), it must first be validated by the security manager before gaining access to these resources. More advance security schemes will be implement by the PaCMAn framework at a later stage.

4 The PaCMAn Framework

The PaCMAn Metacomputer consists of a number of computers located anywhere in the Internet that register with the PaCMAn broker as servers or clients or both. Figure 1 depicts the basic components of the PaCMAn Metacomputer:

- **Broker**: It keeps track of the system configuration. All participating clients and servers and their capabilities are registered with the broker. The broker also monitors the systems resources and performs network traffic/load forecasting [9]. In our system, we have adopted an agent-based version of the Network Weather Service (NWS)[14]. NWS uses CPU and network sensors to create its forecast. The current implementation of NWS provide both a C API and CGI interface. Currently, the PaCMAn broker can utilize NWS without many modifications. The agent daemon process at the server site can start the forecaster whenever it is ready to participate in the PaCMAn Metacomputer. The broker can dispatch special agents that visit these nodes, access the forecast and report it back to the broker.
- **Server**: It is a Java enabled computer (has installed the Java runtime environment) that runs the Tahiti Aglet server, in order to host incoming PaCMAn mobile agents.
- **Client:** It is a Java enabled computer that runs the Tahiti Aglet server, in order to launch/host PaCMAn mobile agents. To be able to launch Aglets from within applets, the Fiji Kit is installed on each client. It includes the FijiApplet, an abstract applet class, which allows the browser to host Aglets, and the Aglet router that must be install at the Web server.

An auxiliary structure of the PaCMAn infrastructure is the Virtual Circular Queue (VCQ) (Figure 2). The VCQ is a distributed circular buffer implemented with Java RMI. It facilitates distributed producer-consumer collaborative work. Portions of the queue reside on different servers that can be distributed in the Internet. Producers of work (JobSlice in our terminology) store it in a portion of the circular queue that is closer to them. If the local portion of the queue is full, the queue server passes the JobSlice to one of its neighbors. Consumers fetch work from the queue. If the local portion of the queue is empty, the local queue server requests a JobSlice from one of its neighbors. The PaCMAn VCQ also supports automatic queue balancing: two neighboring queue servers compare their queue sizes and balance them

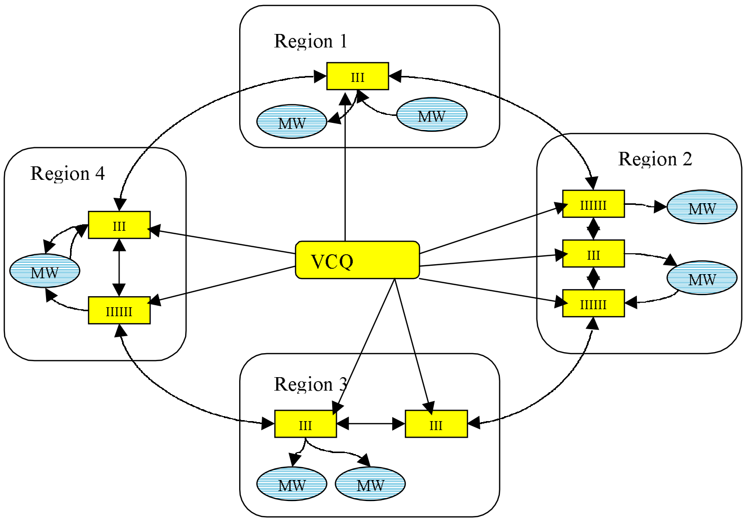


Fig. 2. A example of the VCQ distributed on a number of buffer servers on 4 geographically distant, in the Internet sense, areas

by transferring JobSlices. This procedure is repeated in a round-robin fashion until the entire queue is balanced.

The VCQ due to its static nature is suitable for jobs that have fairly long duration (days, weeks or months) such as the RSA code breaking challenge. For such applications, the participating PaCMAn servers could change very frequently during the lifetime of the application. So the presence of a stationary queue enables the seamless continuation of the work.

To realize our approach, a number of components are defined to complement the existing agent execution environment (i.e., the Aglets). In particular, the existing Aglets are extended with parallel processing capabilities. In addition, a library of task specific objects, called TaskHandlers, is implemented to realize the computation portion of the application. Specifically, the following components comprise the proposed framework:

- **PaCMAn Mobile-Agent**: The main responsibility of the PaCMAn mobile agent is first to create and coordinate other PaCMAn mobile agents and then to initiate a variable number of TaskHandler objects to perform the job specified by its' currently assigned JobSlice. The basic structure of the PaCMAn Agent is depicted in Figure 3. The decision of which object, from the TaskHandler library, will be initiated is done dynamically at runtime based on the current demands, which in turn are determined by the currently assigned JobSlice object.

  The PaCMAn mobile agent can assume various roles in the PaCMAn framework the most common are:
  - **Mobile-Worker (MW)**: This is the workhorse of the PaCMAn system. It performs all the necessary computation, communication and synchro-

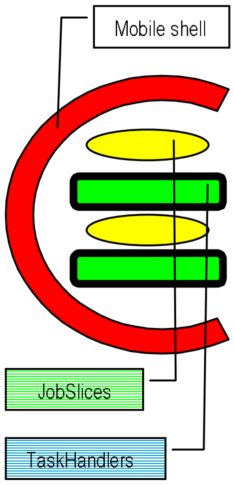
nization tasks. The MW initializes and uses the appropriate TaskHandler. The mode of operation of the MW is similar in concept with the SPMD (Single Program Multiple Data) parallel model: the MW runs in all the servers selected for the application, the same code/process running in all the servers but on different data at each server. The task instructions are passed to it through a JobSlice.

  - **Mobile-Coordinator/Dispatcher(MCD):** The MCD does not participate in the computation. Instead, it performs task-specific coordination among the mobile-workers. This is done by dynamically assigning tasks to the MWs through the use of JobSlices and also by providing information about the status of other MWs. For example, in a computationally intensive application, the MCD can produce a large number of JobSlices (much larger than the number of the MWs), effectively dividing the entire task into a number of subtasks, and dynamic assign a JobSlice to each MW whenever it requests it. This mode of operation is appropriate for application where the load cannot be balanced statically, such as the Prime number generator.

- **TaskHandler Library**: It is a collection of Java objects that are serializable, and thus can travel along with our PaCMAn mobile agents. The TaskHandler is an object that is in the disposal of an agent to use when it needs to perform a specific computational task. For example, when the need arise for a database query, the agent can utilize the DBMS_TaskHandler. The TaskHandler objects are instantiated by the mobile agents in new separate threads that run along with the agents. Initially, the library has only our TaskHandler abstract class. It can be enriched with other TaskHandler objects that must be descendants of our TaskHandler abstract class (this is done both for technical reasons and to ensure a common interface, although it can be overridden). Finally, this library contains an object called the JobSlice object. The JobSlice object holds all the necessary initialization information for the TaskHandler which will be used. JobSlices are essential to the framework, as in effect they act as the main pipeline between our mobile agents and the TaskHandler objects. The only thing that our agent needs to start working on a problem is a JobSlice. When the current JobSlice is finished, the agent can request a new one. The JobSlice is generic and can be used by all the TaskHandler objects of the library.

As stated earlier the VCQ is stationary and is better suited for long running jobs. The MCD on the other hand is used for dynamic agent coordination. The MCD moves if there is a need to provide synchronization to a group of MWs. Of course, it is possible to implement a more dynamic VCQ by linking together a number of MCDs.

Figure 3 depicts some code segments that illustrate the dynamic nature of the JobSlice and the TaskHandler. The mobile shell has a JobSlice and a TaskHandler variables. These are used to reference the JobSlice and TaskHandler ob-



The following method could be used to create and utilize the TaskHandler:

```
JobSlice useTaskHandler (JobSlice job){
   TaskHandler MyMainTask = null;
   try {
        MyMainTask = (TaskHandler)
                   (Class.forName(job.getTH())).newInstance();
   } catch (Exception e) {
        System.out.println(job.getTH());
        System.out.println ("Error while initializing
                   taskhandler:\n" + e.getMessage());
   }
   MyMainTask.init(job);
   new Thread(MyMainTask).start();
   while (MyMainTask.isFinished() != true) { Thread.yield(); }

   return MyMainTask.getResultSlice();
}
```

A JobSlice is used as input and a ResultSlice (technically a JobSlice) is produced by the used TaskHandler. Method getTH of JobSlice returns a string with the name of the class of the TaskHandler to use (Example: frameworks.Pacman.DBMS_TaskHandler).

Fig. 3. The dynamic relationship of the PaCMAn MA, TaskHandler and JobSlice

jects respectivly. The JobSlice object is given to the mobile shell whereas the TaskHandler object is created by the shell. Which class is used as a TaskHandler is determined by the JobSlice object.

5 Object Definitions of the PaCMAn Framework

Mobile worker: As shown in Figure 4, the mobile worker is defined to be a descendant of the Aglet class (in the Figure, this is represented with the PaCMAn Mobile Agent). Besides the functionality inherited from the aglet class, the additional responsibilities of a mobile worker are to:

- Maintain a list of locations and Ids of the other mobile-workers and MCD.
- Create and dispatch other mobile workers.
- Communicate with other mobile workers,
- Communicate with an MCD to get more JobSlices and or instructions about its course of action.
- Instantiate and use TaskHandler objects, either to complete its given task or to handle a message such as joining partial results

In our framework, the mobility and basic communication [2] events are inherited from the Aglets framework and overridden as needed. Due to this fact and the concept of TaskHandler objects that are dynamically assigned, the object definition of the mobile worker strongly resembles the object definition of an Aglet. That is because most of the actions taken by a mobile worker are results

[2] The communication scheme in the Aglet framework is based on event handling. When a message arrives an event is triggered and as a result the handleMessage method is called.

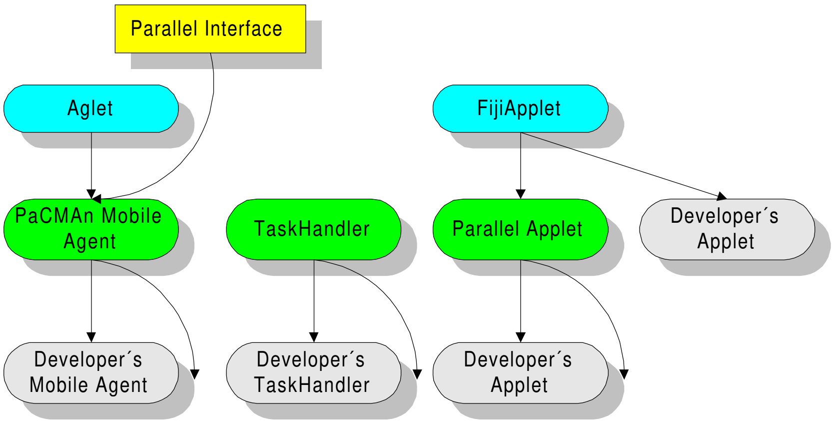


Fig. 4. The agents' hierarchy

of, or have as a result, some form of communication with other members of the PaCMAn framework or some event triggered by the Aglets framework.

In order to send a message from Aglet A to Aglet B, Aglet A must know Aglet's B agletID (a unique identity) and its location. The agletID is not predetermined but is created (by the Tahiti server), when an Aglet is created. Thus, only after the creation of all the PaCMAn mobile agents and only the "leader" PaCMAn mobile agent has access to this information. This makes it necessary to organize this information in a routing table. The routing table contains the URL of each PaCMAn mobile agent, its agletID and its logical ID (a serial number given by our application). At an early point during execution, the leader broadcasts the routing table to all other agents. When a mobile agent moves, the routing table needs to be updated. Broadcasting a message can do this. The departing mobile agent has to leave its proxy on the Tahiti server at its initial location in case messages are in transit while the routing tables are updated.

The Aglets framework provides us with the basic communication methods, for both handling and generating messages:

(i) sendMessage(Message), sendOnewayMessage(Message), sendAsyncMessage(Message), sendFutureMessage(Message) which are responsible for sending messages in a peer to peer scheme.
(ii) getXXXXReply() methods which get a reply that it was sent in response for an older message.
(iii) waitForReply() which waits for a reply from a message sent.
(iv) sendReply(XXXX) methods which send a reply to a newly arrived message.

On the handling side, there is no need for receiving methods (like those in MPI). Message receipt is both implicit and explicit. When a message arrives at an Aglet, it is handled implicitly by the handleMessage(Message) method except in the case an explicitly getXXXXReply() method is used. In the Aglet workbench, there were no broadcast and collective operations, thus in our extension, we included such methods for broadcast and collective operations. Aglets have multicasting facilities but not true broadcasting. Aglets' multicasting works only within a single context [3]. Furthermore, Aglets' multicasting works by letting Aglets register to receive specific type of messages. Since PaCMAn is designed to work with multiple servers and thus different contexts, we had to implement our broadcast operations in the same fashion as MPI broadcast. The same applies to collective operations.

The mobility events provided by the Aglet framework are described with great detail in the Aglet documentation [2]. Apart from these, the run() method is the central to the mobile worker as this is the method responsible for the creation and utilization of the main TaskHandler object(s). All other methods of a mobile worker exist as auxiliary methods of the run() method. TaskHandler objects implement the task specific actions and other needed actions, like reformatting the results (instead of having specific methods of the mobile worker for that purpose). TaskHandlers are dynamically assigned to the run() method. Figure 3 depicts the relationship of the PaCMAn MA, with the JobSlice and TaskHandler objects.

Mobile coordinator/dispatcher: The MCD can be defined either to be a direct descendant of the Aglet class or by refining the mobile worker. This is because the MCD is identical with a mobile worker that uses a TaskHandler to do the coordination and dispatching (again, in Figure 4 this is represented with the PaCMAn Mobile Agent). The basic functionality-responsibilities of a MCD are:

- All the functionality of the mobile worker (in the case where the MCD is its descendant)
- Maintaining and handling (giving away, adding etc.) a list of JobSlices.
- Communicating with other members of the framework.
- Tracking the PaCMAn mobile agents status.

The MCD exists to serve requests for JobSlices and to help coordinating tasks like the combination of partial results. For example, the MCD can be used to balance the computation load required for the production of some partial result, when it is difficult to do so statically. The MCD can also maximize the utilization of our resources. The database application that we use for testing consists of two lines of computations. First, each mobile worker travels to a distributed database and performs its query. Then, the partial results are joined. Thus, some mobile workers will be waiting for others to finish, in order to join their results. The MCD can minimize this waiting time, by instructing the "finished" mobile agents to work together (by exchanging and joining their

ResultSlices, while others are still producing theirs), instead of waiting for a predetermined mobile agent to finish (static scheduling).

Broker: The broker can be defined as a direct descendant of the Aglet class. The basic functionality- responsibilities of a broker are to:

- Maintain a list of available Tahiti servers, as well as network traffic and load information for these servers.
- Maintain a list of the PaCMAn mobile agents currently running. Information about their whereabouts should also be kept.
- Maintain information on useful (and/or needed) resources.
- Communicate with other members of the framework.

The broker's job is to enforce a load balancing mechanism on the PaCMAn framework. This is done by constantly monitoring the activity of the servers. An agent-based version of the Network Weather Forecast [14] system has been adopted for our work because it fulfills all our requirements.

TaskHandler and JobSlice objects: The TaskHandler object defined to be a descendant of the TaskHandler class. The basic functionality-responsibilities of a TaskHandler object are to:

- Be aware of its current state.
- Be implemented as an object that can exist in a thread (in other words to implement Runnable)
- Be able to return a ResultSlice[3] to the Aglet that uses it.
- Accept commands in the form of messages from the PaCMAn mobile agent that owns it.
- Accept new JobSlices from the PaCMAn mobile agent that owns it.
- Carry the code for the application specific task.

The JobSlice object is a Java object with the following functionality-responsibilities:

- Contain the task initialization data.
- Know which TaskHandler can perform the specific JobSlice task.
- Know if it is a normal or a result slice.

[3] A ResultSlice object is technically a JobSlice object created by the processing of other JobSlices.

5.1 Extending the PaCMAn Framework

In the Java agent based technology, an agent can be further viewed as an object that can be inherited, enhanced or refined just like any other object. The Aglets can be viewed as the basic abstract object that provides us with the basic capabilities of mobility, communication, ability to cloning themselves, i.e., implementing all the capabilities supported by the Aglets Framework.

The PaCMAn framework enhances these capabilities with the addition of parallel processing features. We create a hierarchy that can later on be used to create more complex and sophisticated applications. Figure 4 shows the PaCMAn hierarchy. In order to overcome the lack of multiple inheritance in Java, we had to use both interfaces[4] and classes.

To enhance the new Aglets with additional capabilities is now straightforward. For example, to extend the parallel framework with database capabilities we only have to create a database TaskHandler [8], make it a descendant of the TaskHandler class and extend the PaCMAn mobile agent in order to use the DBMS TaskHandler (which could be any of the Developer's TaskHandler in Figure 4). Such a scenario is demonstrated in the section below.

5.2 Implementation Roadmap

The PaCMAn framework provides a great degree of generality. The static part of our agent which is responsible for the mobility, the general coordination, the creation and manipulation of other Agents. The dynamic part deals with the assignment of TaskHandlers. PaCMAn framework a user must follow the following steps:

(i) The client creates a PaCMAn-MW that acts as the master process:
   - Extends the PaCMAn mobile agent class or implements our PaCMAn mobile agent interface[5] or
   - Programs the Parallel-applet either by extending our Parallel-applet class or by writing a new one which extends the IBM's FijiApplet abstract class. (This applet is used as the interface between the user and the PaCMAn mobile agent).

[4] Interfaces are somewhat like abstract classes (they contain method declarations but not code whereas abstract classes may contain code). The difference is that a class can implement as many interfaces as it likes instead of only one, which is the case when extending another class.

[5] We provide this option for compatibility with other frameworks that use IBM's Aglets.

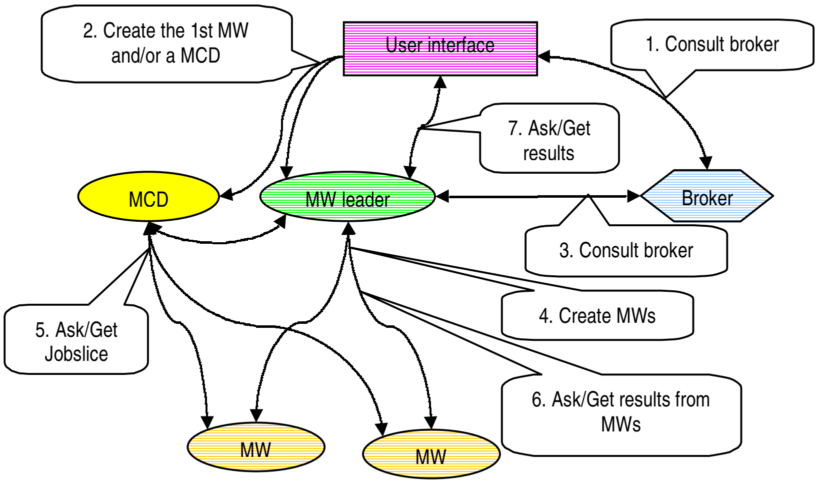


Fig. 5. Execution steps

- Programs the TaskHandler objects by extending our TaskHandler abstract class (thus adding the necessary objects in the TaskHandler library) or by using an existing one from the current TaskHandler library.

(ii) The client machine contacts the broker and requests a network-based forecast. Based on the forecast, the client machine decides on the number and location of the servers it will utilize.

(iii) A number of PaCMAn-MW and PaCMAn-MCD are created and launched.

(iv) The PaCMAn-MWs arrive at their destinations and request from the TaskHandlers to perform specific tasks. While waiting the first TaskHandler to perform its tasks, more TaskHandler objects can be launched to accomplish other tasks as determined by the available JobSlices.

(v) Finally the PaCMAn-MWs co-ordinate among themselves, possibly with the aid of a PaCMAn-MCD, to collect the partial results and returns them to the user.

6 Prototype Implementation and Testing

Our prototype implementation consists of the PaCMAn agents, the TaskHandler library and the PaCMAn GUI. The PaCMAn family agent and the TaskHandler library have been extensively explained in the previous sections. The PaCMAn interface is a generic user interface that is used to start PaCMAn applications. Through this interface the user can input the necessary parameters (for each application), start the application and finally see the produced results. The execution steps taken once the application starts, through the GUI, are the following (Figure 5):

(i) A mobile worker (and possibly a MCD) is created. This first Mobile worker acts as the leader. Note that when a MCD is not created the user interface provides the leader with all the necessary JobSlices (for all the MWs). In the case that a MCD is used, only the MCD is provided with a JobSlice. From that point on the MWs get their JobSlices from the MCD.

(ii) If necessary creates and fires other mobile workers and MCDs (passing the appropriate JobSlices to them).

(iii) The mobile workers dispatch themselves to the appropriate server. Note all mobile workers other than the leader start from this step.

(iv) Each MW creates a TaskHandler object and initiates it in new separate thread. Which TaskHandler will be loaded is determined from the JobSlice currently assigned to the MW. To complete the TaskHandler initialization the MW passes its' JobSlice to the TaskHandler.

(v) The TaskHandler perform its task concurrently with the mobile worker.

(vi) The mobile worker requests from the TaskHandler to perform a specific task. While waiting the first TaskHandler to performed its tasks more TaskHandler objects can be launched to accomplish other tasks.

(vii) Finally the mobile worker collects, in coordination with the other mobile workers and under the guidance of the MCD, the results (the produced ResultSlices) and returns them to the user. In other words the results are channeled through various MWs back to the user. If necessary more TaskHandlers are created and utilized to join the results (just like the database application case).

To evaluate the performance of PaCMAn prototype we developed some sample applications, which were tested and analyzed. These applications demonstrate all of the PaCMAn framework components with the exception of the broker and the VCQ that have not been fully integrated in the prototype yet. The test suite used for the performance analysis was made up with following applications:

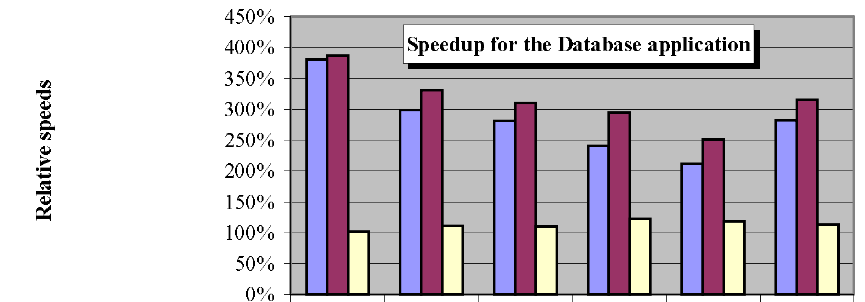
(i) **Parallel database querying over the network[10]**. This is a medium computation load application that is split in two parts. This application is a database application in which the client wants to retrieve data that are distributed in a number of different servers. Thus the actual problem is first to retrieve the data and then join them as needed.

(ii) **Computing prime numbers.** This is a heavy computation load application that is difficult to load-balance statically: It counts the number of prime numbers in a given range.

(iii) **The trapezoidal rule:** This is a light computation load application: an implementation of the classic trapezoidal rule problem.

(iv) **Cracking the RC5 algorithm**[12]. This is a medium computation load application: it searches for the decryption key in a given range.

For the implementation of the above applications, we just need to create the appropriate TaskHandlers that would perform the given tasks:



Sample size	200	400	600	800	1000	Average
Speedup from mode 0 to 1	380.7%	299.1%	281.1%	240.9%	211.9%	282.7%
Speedup from mode 0 to 2	386.7%	331.1%	310.5%	295.1%	251.6%	315.0%
Speedup from mode 1 to 2	101.6%	110.7%	110.5%	122.5%	118.7%	112.8%

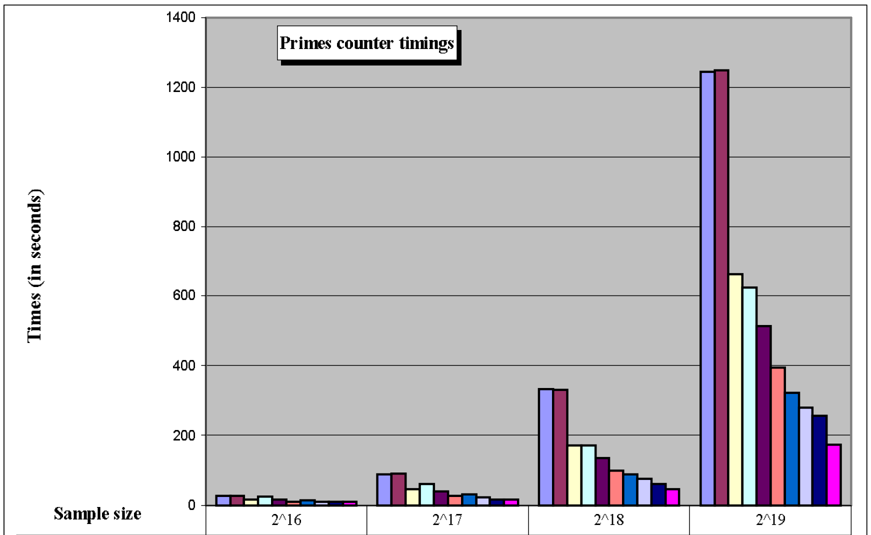
Fig. 6. Database application speedups

(i) **DBMS_TaskHandler**, which performs the task of the data connection and retrieval.
(ii) **Join_TaskHandler**, which joins the results produced by the DBMS_TaskHandler.
(iii) **Primes_TaskHandler**, which finds the prime numbers in a given range.
(iv) **Trap_TaskHandler**, which calculates the area of a trapezoid.
(v) **RC5_TaskHandler**, which finds the key (if it exists) in a given range.

Each of these TaskHandlers is an autonomous Java object and can exist outside of the scope of the PaCMAn framework. For example the code of the Primes_TaskHandler was taken from another application and it was easily adapted to the PaCMAn framework. This is also the case for the RC5_TaskHandler that was provided to us by the authors of [12]. We tested the above applications with a group of 8 servers. Furthermore there were the following test modes:

Mode 0: One agent moving from server to server gathering and joining all the results (applicable only to the database application).
Mode 1: N agents on N servers and every agent sends its results to one (leader) agent.
Mode 2: N agents on N servers and the produced results are joined in a tree reduction order.

The database application: This is web-database query application. The data is distributed on a number of databases residing on different servers. Mobile workers are launched to perform queries independently and then co-operated to join the results of the individual queries. The speedups for this experiment are tabulated in Figure 6. Sample size refers to the number of records returned by each query. The speedup of mode 1 and 2 are 282 and 315% compare with mode 0. The reason we achieved such high speedups is that mode 0 simulates a serial execution while mode 1 and 2 are parallel



Sample size	2^16	2^17	2^18	2^19
1 host	26.0	87.9	332.0	1244.3
2 hosts, unbalanced	26.7	90.3	330.9	1247.9
2 hosts, statically balanced	15.7	46.8	171.5	663.4
2 hosts, MCD mode	24.2	61.6	171.7	623.9
4 hosts, unbalanced	15.3	40.3	136.2	514.9
4 hosts, statically balanced	10.3	26.0	98.2	395.7
4 hosts, MCD mode	12.9	31.4	89.8	321.2
8 hosts, unbalanced	10.1	21.8	75.3	279.7
8 hosts, statically balanced	10.1	15.3	60.6	256.6
8 hosts, MCD mode	10.3	15.5	47.1	173.6

Fig. 7. Primes generator timings

implementations. The improvement (1-10%) between modes 1 and 2 is due to the fact that in mode 2 we have more parallelism during the joining phase. We expect higher speedups of mode 2 compare to mode 1 in application were the joining of the partial results requires heavy duty computation.

The prime generator: It is very difficult to have static load balancing in applications such as the prime number generator. Thus, we have experimented with 3 different implementations. In the first case, there was no load balancing: the entire range is divided into as many equal parts as Mobile Agents used. In the second case, we statically load balanced the application while in the third case we have dynamic load balancing through the use of an MCD. Under this implementation we have divided the range into a large number of JobSlices and loaded them on the MCD. Whenever a Mobile Agent finishes the range specified by its JobSlice it request a new JobSlice from the MCD. The test parameters for this application are the number of servers and the sample size (the largest number that is going to be checked). We tested this application for problem sizes $2^{16} - 2^{19}$. The execution timings from the experiments are summarized in Figure 7. The relative speedups are summarized in Figure 8. The speedups reported are calculated with respect to the execution time of with only one Mobile agent in one server.

		2 hosts vs 1		
Sample:	**2^16**	**2^17**	**2^18**	**2^19**
Balanced	165.9%	187.9%	193.6%	187.6%
Unbalanced	97.4%	97.4%	100.3%	99.7%
MCD	107.3%	142.8%	193.4%	199.4%

		4 hosts vs 1		
Sample:	**2^16**	**2^17**	**2^18**	**2^19**
Balanced	251.9%	338.5%	338.1%	314.5%
Unbalanced	169.4%	218.3%	243.8%	241.6%
MCD	201.6%	280.0%	369.7%	387.5%

		8 hosts vs 1		
Sample:	**2^16**	**2^17**	**2^18**	**2^19**
Balanced	257.9%	576.5%	547.9%	484.9%
Unbalanced	255.9%	403.9%	441.1%	444.9%
MCD	251.9%	565.9%	705.3%	716.7%

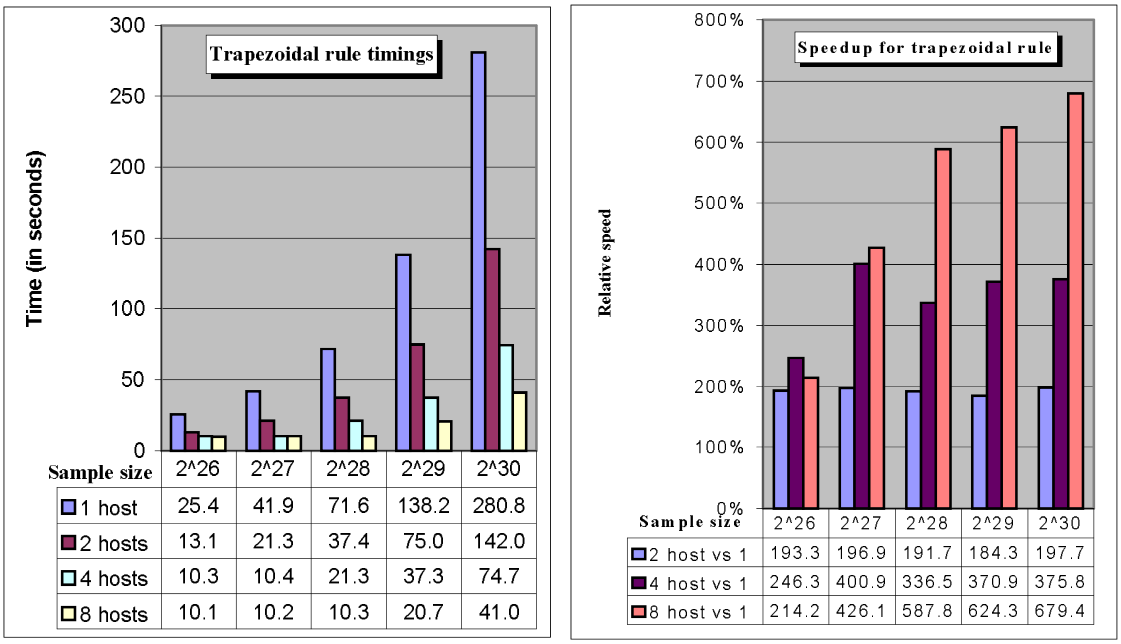
Fig. 8. Primes generator speedups

Figure 8 demonstrates as expected that the better the load balancing is, the more efficient the parallel processing is. It worth noting that as the problem becomes larger the dynamic load balancing performed by the MCD becomes more efficient. This somewhat unexpected behavior is because our static load balancing did not achieve the absolute perfect balance. On a closer examination, we have concluded that the MCD provides better performance as the number of servers increases. This is due to the fact that even with identical machines you can never get the exact same throughput. This becomes noticeable only with sufficiently large problems where even the slightest difference in performance has a great impact.

The trapezoidal rule: The results from the trapezoidal rule tests are depicted in Figure 9. Test parameters here are the number of servers and the sample size. The sample size refers to the number of the trapezoidals. The speedups and the parallel efficiency achieved are very respectable.

The RC5 cracker: Figure 10 depicts the test results for the RC5 cracker. The test parameters are the number of the servers and the sample size. The sample size refers to the length of the range that is searched. The results show very good, almost linear, speedups as was the case for the trapezoidal rule.

Overall the experimentation with the prototype showed that the PaCMAn approach provides good parallel efficiency ranging from 79 to 88 % over the entire test suite. These are very respectable figures for a network based Metacomputer. The results for using mode 2 did not appear to improve performance in any application other than the database one. Mode 2 is suitable for large number of workers and/or when there is heavy computation requirements in



Sample size	2^26	2^27	2^28	2^29	2^30
1 host	25.4	41.9	71.6	138.2	280.8
2 hosts	13.1	21.3	37.4	75.0	142.0
4 hosts	10.3	10.4	21.3	37.3	74.7
8 hosts	10.1	10.2	10.3	20.7	41.0

Sample size	2^26	2^27	2^28	2^29	2^30
2 host vs 1	193.3	196.9	191.7	184.3	197.7
4 host vs 1	246.3	400.9	336.5	370.9	375.8
8 host vs 1	214.2	426.1	587.8	624.3	679.4

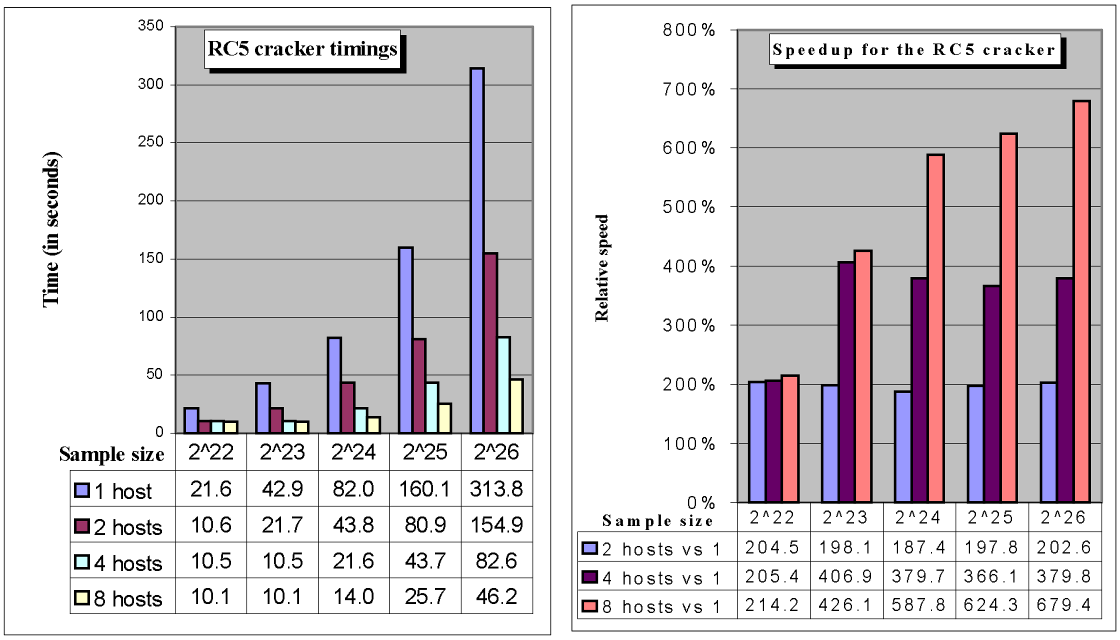
Fig. 9. Trapezoidal rule timings and speedups

the joining phase of the partial results. We will be able to test this mode more rigorously with the UNIX port of PaCMAn

7 Related Work

A number of Java-based systems have been proposed for Metacomputing. Although the goal is more or less the same, to utilize heterogeneous hosts for High Performance Computing (HPC), the approaches differ. In this section we provide a brief overview of systems that have similar goals and/or approach with PaCMAn.

The systems that are more closely related to PaCMAn are the volunteer-based systems. The basic idea is to have a collection of applets that communicates with each other to solve problem distributed. This is possible with the use of brokers that routes the messages between the applets. Example of such systems are Javelin [15], Charlotte [22] and Bayanihan [23]. The common ground of these systems and PaCMAn is the use mobile code, applets and Mobile Agents respectively to solve a problem in parallel. The applet-based approach is passive, it relies on servers to volunteer, each time their participation, by pointing to the systems WWW page. PaCMAn on the other hand is proactive. After a server is registered it can be used without any need for collaboration from the owner of the server. A performance drawback of the applet-based is that all communication has to be routed through the broker. Furthermore, these systems are browser-centric so they have to incur all the extra security



Sample size	2^22	2^23	2^24	2^25	2^26
1 host	21.6	42.9	82.0	160.1	313.8
2 hosts	10.6	21.7	43.8	80.9	154.9
4 hosts	10.5	10.5	21.6	43.7	82.6
8 hosts	10.1	10.1	14.0	25.7	46.2

Sample size	2^22	2^23	2^24	2^25	2^26
2 hosts vs 1	204.5	198.1	187.4	197.8	202.6
4 hosts vs 1	205.4	406.9	379.7	366.1	379.8
8 hosts vs 1	214.2	426.1	587.8	624.3	679.4

Fig. 10. RC5 cracker timings and speedups

restrictions imposed by the browsers.

A different approach is to implement in Java the functionality of classical parallel processing and coordination systems systems such as PVM. The JPVM library [21] implements in Java the basic functionality of the C and Fortran interface of PVM. The mobility of our system makes it more dynamic and puts it in a better position to utilize the Internet for Metacomputing that these adaptations of such "legacy" systems.

New Java dialects and extensions of the Java Virtual Machine (JVM) is another research direction for utilizing Java for HPC. One such project is the Java/DSM (Distributed Share Memory) [16] which extends the JVM to provide a shared memory abstraction over a group of physically distributed machines, thus creating a Metacomputer which is similar in concept with the PaCMAn Metacomputer. PaCMAn is a more dynamic system that utilizes mobility and uses the message passing paradigm. Other dialects and extensions of Java (expansion by implementing a compiler) provide parallel execution models and constructs. Titanium [17], a Java dialect, is one of them. A similar approach is the use of Java bytecode optimizers that discover implicit parallelism, like javab [18]. One very interesting approach is the automated generation of binding for native libraries such as MPI [19]. The last two approaches are of interest to us because they could be used in conjunction with PaCMAn. Such a combination will provide an optimized bytecode that runs faster on the lowest layer were the PaCMAn Metacomputer will operate on the higher layer.

Acknowledgments

The authors wish to thank Stavros Papastavros for his contributions in the earlier system for Parallel processing with Java Mobile Agents [27] that we developed and its valuable input during the development of PaCMAn.

8 Concluding Remarks

In this paper we have presented the PaCMAn system, an Internet based Metacomputer that can utilize heterogeneous resources to tackle large problems in parallel. The PaCMAn Metacomputer has three basic components: Servers, Brokers and Clients. Clients launch multiple Java-mobile Agents that roam around the Internet and visit registered Servers to perform their task in parallel by communicating and cooperating. The Brokers/load forecasters keep track of the available resources and provide load forecast to the Clients. Our system is modular and dynamic. The mobile shell is separated from the application code. The application code is implemented as TaskHandler objects which are implicitly loaded to our Mobile shell. Thus, the end-user of our system does have to deal with the underlining primitives that support mobility and coordination. The transformation of a Java Applet to a PaCMAn TaskHandler is a straight forward process and we are planning to have it automated.

An application that is implemented accordingly to the PaCMAn framework is boosted by the inherited benefits of using Java mobile agents and the WWW interface. An important advantage is that the number of processes can change dynamically. The servers that will host the PaCMAn-MW can be decided at run time, after considering the network traffic or the workload of the available servers. PaCMAn-MW can be reused as they can receive new instructions and then use new tasks i.e., initiate different TaskHandler objects according to its instructions. Furthermore, the servers that host worker processes are not required to be part of the same local network with the client. A server can be located anywhere in the entire Web. A PaCMAn-MW can move to servers where the available resources and environment are suitable for its kind of work. Our system works equally well in wireline and wireless networks. PaCMAn-MWs can be launched (and forgotten) from a laptop or a palmtop computer during a short Web session. The PaCMAn-MW can roam around the unstructured network to perform its tasks and then wait until the communication link is again available to return home with their task results.

We have developed three versions of PaCMAn two of them are based on the Aglet framework and one on the Voyager. The prototype used for experimentation presented here is using the aglets and it is PC-based. The Unix version that uses aglet and the Voyager version are currently been tested. The experimental results from our prototype showed that PaCMAn systems can have very good parallel efficiency. Our future plans is use PaCMAn for distributed data-mining and also develop versions of PaCMAn that utilize native libraries.

References

[1] Evaggelia Pitoura and George Samaras, *"Data Management for Mobile Computing"*, Kluwer Academic Publishers, 1997.

[2] *"Aglets Workbench"*, by IBM Japan Research Group. Web site: http://aglets.trl.ibm.co.jp

[3] Danny B. Lange and Mitsuru Oshima *"Programming and Deploying Java Mobile Agents with Aglets"*, ISBN: 0-201-3258-9.

[4] D. Chess, B. Grosof, C. Harrison, D.Levine, C. Parris, G. Tsudik. *"Itinerant Agents for Mobile Computing"*, Journal IEEE Personal Communications, Vol. 2, No. 5, October 1993.

[5] J. E. White, *"Mobile Agents. General Magic White Paper"*. Web site http://www.genmagic.com/agents, 1996.

[6] Jeff Nelson, *"Programming Mobile Objects with Java"*, John Wiley & Sons, ISBN: 0471254061.

[7] George Samaras, Marios D. Dikaiakos, Constantinos Spyrou, Andreas Liverdos, *"Mobile Agent Platforms for Web Databases: A Qualitative and Quantitive Assessment"*, Proceedings of first International Symposium on Agent Systems and Applications and third International Symposium on Mobile Agents (ASA/MA 99), October 1999, Palm Springs, California.

[8] S. Papastavrou, G. Samaras, and E. Pitoura, *"Mobile Agents for WWW Distributed Database Access"*, Proceedings of 15th International Data Engineering Conference, Sydney, Australia, March 1999.

[9] D. Barelos, E. Pitoura, G. Samaras, *"Mobile Agents Procedures: Metacomputing in Java"*, Proceedings of the ICDCS Workshop on Distributed Middleware, (in conduction with the 19th IEEE International Conference on Distributed Computing Systems (ICDCS99)) Austin, TX USA, June 1999.

[10] C. Panayiotou, G. Samaras, E. Pitoura and P. Evripidou, *"Parallel Computing Using Java Mobile Agents"*, Proceedings of 25th Euromicro Conference, In Press, Milan, Italy September 1999.

[11] Paraskevas Evripidou, George Samaras, Evaggelia Pitoura and Christoforos Panayiotou, *"PaCMAn Mobility for High Performance Computing"*, Workshop on Java for High-Performance Computing, In conjunction with the 1999 ACM International Conference on Supercomputing, June 1999, Rhodes, Greece.

[12] A. Patel, P. Gladychev, D. Omahony. *"Cracking RC5 with Java applets"*, Concurrency Practice and Experience, Vol 10 (11-13), 1165-1171 (1998).

[13] L. Bic, M. Dillencourt, M. Fukuda. *"Mobile Agents, Coordination and Self-Migrating Threads: A common Framework"*. http//:www.ics.uci.edu/bic/messengers/messengers.htm.

[14] R. Wolski, N.T. Spring and J. Hayes, *"The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing"*. Journal of Future Generation Computing, 1999, In press.

[15] Bernd 0. Christiansen et al. *"Javelin: Internet-Based Parallel Computing Using Java"*, ACM Workshop on Java for Science and Engineering Computation, Las Vegas, NV, June 21, 1997.

[16] Weimin Yu and Alan Cox. *"Java/DSM: A Platform for Heterogeneous Computing"*, ACM Workshop on Java for Science and Engineering Computation, July 1997

[17] Kathy Yelick, et al. *"Titanium: A High-Performance Java Dialect"*, ACM Workshop on Java for High-Performance Network Computing, Stanford, CA, Febr. 1998.

[18] Aart J.C. Bik and Dennis B. Cannon. *"javab Manual"*, ACM 1998 .Workshop on Java for High-Performance Network Computing, Stanford, CA, Feb. 1998.

[19] Vladimir Getov, Susan Flynn-Hummel, Sava Mintchev. *"High-Performance Parallel Programming in Java: Exploiting Native Libraries"*, ACM Workshop on Java for High-Performance Network Computing, Stanford, CA, Feb. 1998.

[20] K. Mani Chandy et al. *"A World-Wide Distributed System Using Java and the Internet"*. IEEE International Symposium on High Performance Distributed Computing. (HPDC-5), Syracuse, New York, August 1996

[21] Adam J. Ferrari. *"JPVM: Network Parallel Computing in Java"*, ACM Workshop on Java for High-Performance Network Computing, Stanford CA, Feb. 1998.

[22] A. Bartloo, et Al., *"Charlotte: Metacomputing on the Web"*, in Proceedings of 9th International Conference on PDCS, 1996.

[23] L. Sarmenta and S. Hirano, *"Bayanihan: Building and studying web-based volunteer computing systems using Java"*, Future Generation of Computer Systems, Vol.15(6), 1999.

[24] DESCHALL. *"Internet-Linked Computer Challenge Data Encryption Standard"*. Press Release, 1997.

[25] *"Voyager"*, by ObjectSpace. Web site: http://www.objectspace.com/

[26] *"Java Grande Forum"*, Web site: http://www.javagrande.org/

[27] S. Papastavros, G. Samaras and P. Evripidou, *"Parallel Web Querying using Java Mobile Agents and Threads"*. Technical Report TR-98-7, Department of Computer Science, University of Cyprus, May 1998.