Afro-Asian Journal of Scientific Research (AAJSR)
المجلة الأفروآسيوية للبحث العلمي (AAJSR)
E-ISSN: 2959-6505
Volume 1, Issue 3, July-September 2023, Page No: 204-214
Website: https://www.aajsr.com
SJIFactor 2023: 3.908 **ISI 2023: 0.337**

Mobile Agent Systems: Execution Code on a Mobile Agent vs Execution Code on a Stationary Agent

Atiya Alsnousi Ahmida[1*], Saad B. Mathe[2]
[1,2] Department of Computer Technology, Faculty of Technical Sciences, Sebha, Libya

*Corresponding author: atiya-auhida@ctss.edu.ly

Received: July 14, 2023	Accepted: August 19, 2023	Published: August 31, 2023

Abstract:
While many researchers investigated the merits of the mobile agent paradigm over the conventional methods like Client/Server and Code on demand, very few provided insight into the software agents community itself. Such an insight is provided in this paper. Within the context of the software agents, two categories can be identified: stationary agents that execute only on the system where they begin execution (Home) and mobile Agents that are not bound to the system where they begin execution. A mobile agent is first residing on a home machine, and it is dispatched to a remote host for execution. Like any other computer program, a software agent needs code to execute. This code could be installed on the stationary agent at the server side or it could be pre-imbued to the mobile agent from the client side. The placement of the execution code is the main objective of this paper. Using Aglet Software Development kit (ASDK) two states are studied, first where the execution code is pre-imbued into the mobile agent and second where the execution code is preinstalled on the remote server and manipulated by the stationary agent on that server. In terms of performance, the single mobile agent is compared to a stationary agent, and multiple mobile agents are compared to a stationary agent.

Keywords: Mobile Agent 1, Stationary Agent 2, Software Agent 3 Execute Code, 4 ASDK 5

Cite this article as: A. A. Ahmida, S. B. Mathe2, "Mobile Agent Systems: Execution Code on a Mobile Agent vs Execution Code on a Stationary Agent," *Afro-Asian Journal of Scientific Research (AAJSR),* vol. 1, no. 3, pp. 204–214, July-September 2023.

نظام الوكيل المتحرك: الشيفرة التنفيذ علي الوكيل المتحرك مقارنة مع الشيفرة التنفيذ على الوكيل الثابت

عطية السنوسي أحميدة[1*]، سعد يشير ماضي[2]
[1،2] قسم تقنيات الحاسوب، كلية العلوم التقنية، سبها، ليبيا

الملخص
أن العديد من الباحثين في نموذج الوكيل البرمجي software agent paradigm يحققون في مزايا هذا النموذج المتنقل مقارنة بالأليات التقليدية مثل العميل / الخادم Client/Server وشفير عند الطلب Demand Code On، وكذلك نظام التوازي المعروف Parallel Virtual Machin PVM، قليل منهم أبدي اهتمام بما في داخل نظام الوكيل البرمجي Software Agent System نفسه. هده الورقة تقدم نظرة من داخل سياق وكلاء البرمجيات Software Agent

Community، التي يمكن تحديد فئتين: العوامل الثابت Stationary Agentsهي البرمجيات التي يتم تنفيذ مهامها فقط على النظام التي أنشئت عليها وتسمي (Home)، أما الفئة الثانية فهي الوكيل المتنقل mobile Agents.
وهذا النوع غير مقيد بالنظام أو الأجهزة التي أنشاء عليها وهو يعمل على شبكة واسعة من الأجهزة المضيفة Home Machine. وهو برنامج ينشأ ويتم إرساله إلى مضيف البعيد لتنفيذ عمليات معينة مثل أي برنامج كمبيوتر آخر، يحتاج الوكيل الي شفيرة برمجية لتنفيذ هده المهام. حيث يمكن تثبيت هذا الشيفرة أو الرمز على الوكيل الثابت في جهاز المضيف أو يمكن تثبيته مسبقًا ضمن الوكيل المتحرك من جانب العميل. إن وضع الشيفرة أو رمز التنفيذ هو الهدف الرئيسي لهذه الورقة البحثية باستخدام Agent Software Development Kit (ASDK) حالتان يتم دراستها في هده الورقة: الحالة الولي عندما تكون شفيرة التنفيذ مشحونة مسبقًا ضمن الوكيل المتنقل ويتم تشغيلها ومعالجته على الأجهزة المضيفة بواسطة الوكيل الثابت على ذلك المضيف والحالة الثانية حيث يتم تثبيت شيفرة التنفيذ مسبقًا على الوكيل الثابت في جهاز المضيف ويتم معالجته بواسطة الوكيل الثابت على ذلك الجهاز ويتحصل الجهاز المضيف علي النتائج فقط. المقارنة تتم هنا من حيث الأداء performance متمثلاً في زمن تنفيذ المهام من حيث الأداء، وتتم مقارنة وكيل متحرك واحد بوكيل ثابت، وتتم مقارنة العديد من الوكلاء المتنقلين بوكيل ثابت.

الكلمات المفتاحية: الوكيل المتنقل، الوكيل الثابت، وكلاء البرمجيات، الشيفرة التنفيذ، حزمة تطوير البرمجيات.

Introduction

History of code mobility

Code mobility is not a new concept. It has its origins in the 1960s when the Job Control Language (JCL) [1]. was used in the optimization of networked computers, enabling minicomputers to submit batch jobs to mainframes using a remote entry system (RES) [2]. Another direct approach to code mobility is the RSH (Remote Shell) command that was introduced by 4.2BSD UNIX in 1984 [3]. It allows a user to send a shell script to a remote machine where it will be executed with access to all the local resources of the remote machine, (such as a printer or data on the local disk of the remote machine).

A rather prominent and successful example for the use of mobile code is SQL, the structured query language [5], which was originally conceived in the late 1970s. Even though the mobility of a particular query was not in the center of the design of SQL, but rather the declarative, high-level, application neutral formulation of a query, the compact representation of the interpreted language makes it very easy to package a query into a message, send it to a DB server, where it is interpreted with complete access to all the resources of the DB server. Thus, it can perform a task on the large amounts of data in the DB and return the (usually) small result of the query to the user.

A major obstacle for truly mobile code that can potentially be executed everywhere, is the heterogeneity of the possible execution environments (hardware, operating system, and installed software). This has led to the development of scripting languages that were conceived to overcome these problems, such as Perl [4]. Tcl [5]. These scripting languages are executed in the context of a runtime system that interprets the commands of the scripting language. Since the runtime system itself is software, it can be ported to any suitable hardware platform, which can then execute mobile code of the corresponding scripting language, thus providing a homogeneous execution environment for the programs coded in the scripting language. All of these scripting languages have to some extent been augmented with mobile code facilities (agent Perl [8], Agent Tcl [6]. etc.). The most recent contenders in the area of mobile code systems are Telescript3 [7] and Java.

Java was originally conceived as an interpreted, architecture-neutral, portable language. However, it was soon discovered that these properties made it into a suitable language for automatically downloadable programs on the World Wide Web. With the integration of the Java virtual machine into the Netscape Navigator in October 1995 it became the first widely deployed system for mobile code. A considerable number of systems try to leverage the code mobility of Java into a full-fledged mobile agent system, such as Aglets, Grasshopper, Mole, Odyssey, Voyager, and many others [8].

Mobile agent paradigm.

Over the conventional methods Software agents further evolve by two more elements: one is client customization; the other is further assembling of the software modules into a self-contained entity. This is different from the conventional approaches, where the software modules are maintained on the server side, and are kept as functions or objects in loosely coupled settings. Software Agents are programs, typically written in script languages, enabled with certain properties to work on behalf of human users in a distributed heterogeneous environment. Software Agents can be either stationary Agents that execute only on the system where they begin execution (Home) or mobile Agents that are not bound to the system where they begin execution. A mobile agent is firstly residing on a home machine, and it is

dispatched to a remote host for execution. The accommodating host would provide suitable runtime environment for the piece of software, the mobile agent, to execute. The mobile agent would execute, collect host-specific information, and generate runtime states and variables ready to migrate to the second host in the itinerary. This process continues until the Mobile Agent returns home with useful information from the last host in the itinerary.

The aglet models.
Major existing paradigms for building distributed applications can be classified into two groups. Examples of the first class include traditional RPC and most recently its object cousins RMI and COBRA. For this class of paradigms, the functionality of applications is partitioned among participating nodes. Different participants use message-passing to coordinate distributed computation. Computation itself is partitioned; participants exchange intermediate results and other synchronization information. For the second class of paradigms, computation is migrated toward resources. This type of paradigm is especially useful for applications requesting immediate reactions to incoming streams of real-time data and distributed applications that are very tightly coupled. In this project, we experiment with one such paradigm—It is the Aglet model. Aglet is the shorthand for agent plus applet. It provides us an infrastructure for building second-class distributed applications.

There are at least three ways to view the role of aglet technology:

- As a communication mechanism
- As a data transport vehicle among hosts
- As a framework for partitioning application functionality

Paper contribution and Objective
The primary goal of this paper is to deliver a measurable performance comparison between a mobile agent and a stationary agent. In order to verify this goal, three applications have been chosen as an arena for this comparison: matrix multiplication, remote document retrieval and remote archive update. The execution time is set to be the determinant.

- **Matrix multiplication operation**: In this application, there are two matrices the elements of the two of them are at the server side. The elements of matrix one has to be retrieved from their respective file and then the elements of matrix two have to be retrieved from their respective file as well then on the remote server the process of multiplying them would be executed finally the result to be send back to the originator on the client machine.
- **Remote document retrieval**: in this application there is a document held by the server and requested to be on the client machine. So, the task here is to retrieve the document and present it on the client machine.
- **Archive update**: In this application there are scattered pieces of information in different files on the remote server. The task is to accumulate them in a particular file named as an Archive file

The comparison falls into two categories:
- Single mobile agent to a stationary agent.
- Multiple mobile agents to a stationary agent.

Single mobile Agent to a stationary agent.
In this context, a single code-holding mobile agent is compared to a code-holding stationary agent. The point of comparison is the time required by any of them to accomplish a set of tasks. In order to do so, an aglet mobile and stationary version are constructed. The mobile version is created on the client machine and dispatched to the remote server by another stationary agent on that machine. The stationary version is activated on the remote host.

- **Execution code on the mobile agent.**

In this case, all codes necessary to execute the tasks on the remote server are put on the mobile agent. Upon arriving at the destination, the incoming Agent will start carrying out the execution of the tasks. The time required to execute each task and the total execution time is measured for the mobile version. Upon finishing the mobile agent would send the results back to the master agent on the client machine.

- **Execution code on the Stationary agent.**

In this case, all codes necessary to execute the tasks on the remote server are held by the stationary agent. Upon arriving at the destination, the mobile Agent would send requests for services to the stationary agent. The stationary agent would receive and process the requests of the visitor and give back the results of execution to the mobile agent which resides on the same server then the mobile agent would send the results to the originator on the client machine. The time required to execute each task and the total execution time is measured for the stationary version.

Multiple-mobile agents to a stationary agent.

In this context, multiple code-holding mobile agents are compared to a code-holding stationary agent. The point of comparison is the time required to accomplish a set of tasks brought by the almost concurrently arriving mobile agents in other words examining how capable a code-holding stationary agent of executing multiple tasks against multiple code-holding mobile agents operating on the same server simultaneously. In order to do so, multiple mobile agents would be created and simultaneously dispatched to the remote host. On the other hand; a stationary agent would be activated on the remote host.

- **Execution code on the mobile agents.**

In this case, all codes necessary to execute the tasks on the remote server are put on the mobile agents. Each mobile agent would be assigned a particular task. Upon arriving at the destination, the incoming Agents would start executing their tasks individually and simultaneously on the remote host. The time required to execute each task is measured individually for mobile agents. Upon finishing mobile agents would separately send the results to their respective originators.

- **Execution code on the Stationary agent.**

In this case, all codes necessary to execute the tasks on the remote server are put on the stationary agent. Each mobile agent would be assigned a request for a particular service from the stationary agent. Upon arriving at the destination mobile Agents would send requests for services to the stationary agent. The stationary agent synchronously would receive and process the requests of the visitors and send the execution results back to them then mobile agents would send the results to their respective originators. The time required to get the service accomplished is measured for each mobile agent.

Implementation results and discussions.

Implementation infrastructure.

The experiments were conducted on a wide area network infrastructure (internet), and two servers were deployed. One was attached to the Jarring Internet Access provider. The other one was attached to TMnet Internet Access provider.

Hardware components:

The basic hardware components consisted of two PCs with the following features:

- Pentium III Processor (733 MHz)
- 128 Mb RAM
- Internet Access

Software components:

The basic software components consisted of the following elements:

Table1: Software basic components

SN.	Software	Version
1	Windows Operating System	Win98 SE
2	Java development kit (JDK)	1.1.8
3	Aglet Software Development kit (ASDK)	1.1.0

Implementation results.

General description.

As it has already been mentioned in the previous chapter three operations would be conducted in different sizes. Table 2 shows the details of the operations.

Table 2: Operations description.

No.	Operation	Size
1.	Matrix multiplication1	120(columns) x 120(rows)
2.	File contents1	250 KB data
3.	File copy1	800 KB data
4	Matrix multiplication2	100(columns) x 100(rows)
5	File contents2	210 KB data
6	File copy2	600 KB data
7	Matrix multiplication3	80(columns) x 80(rows)
8	File contents3	160 KB data
9	File copy3	400 data
10	Matrix multiplication4	70(columns) x 70(rows)

The reason why each operation is conducted in different sizes is to examine whether the size of the operation has an impact on the performance of any of the competitors.

Throughout this part, each operation is denoted with its serial number as appears in Table 2. Time in all cases is measured in milliseconds.

A Single Mobile agent vs a Stationary agent.
In this category, there was a single mobile agent and a stationary agent.
First, a mobile agent without source code was dispatched to the remote host with requests for services from the stationary agent on the remote host. Upon arriving, the mobile agent submitted the requests to the stationary agent. The stationary agent accessed the data sources and carried out the requested services (these were retrieving elements of the matrices and multiplying them, reading the document and updating the archive) then it gave back the final results to the mobile agent. The mobile agent in this case neither bore source code nor had access to the data source on the remote host.
Second, a mobile agent with source code was dispatched to the remote host. The Mobile agent accessed the data source and carried out the execution. The stationary agent in this case neither bore a source code nor accessed the data source on the remote host.

The main purpose here was to see which nimbler to execute the operations. That was determined by the time measured for each of them.

After conducting a series of trials, the average time for both the stationary agent and the mobile agent is presented in Table 3.

Table 3: Mobile & Stationary Real Time

Operations Sequence	Stationary Time	Mobile Time
Operation 1	52262	54974
Operation 2	13738	15803
Operation 3	42231	42533
Operation 4	23976	23832
Operation 5	10214	11634
Operation 6	31343	32582
Operation 7	8650	9034
Operation 8	7717	7965
Operation 9	21405	21377
Operation 10	5037	4943
Total Time	216573	224677

The results of Table 3 are charted in figures (1 and 2):

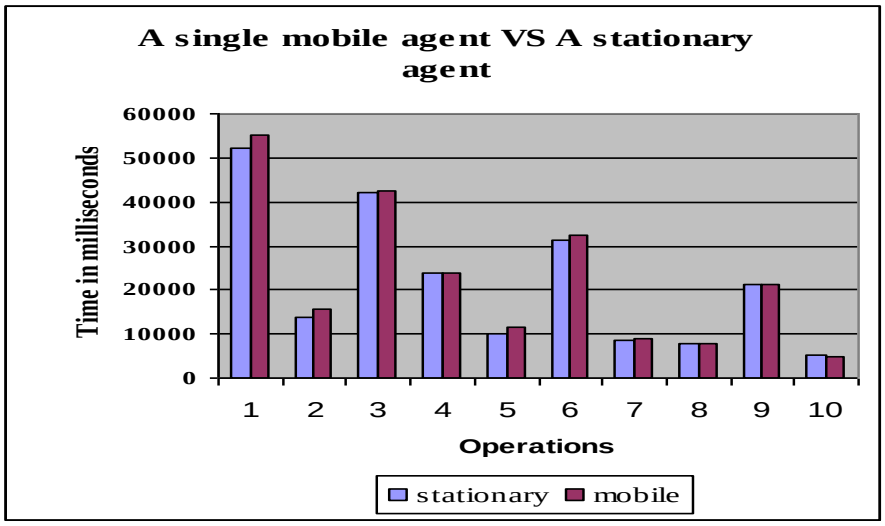

Figure 1: A single Mobile agent vs a stationary agent (individual operations).

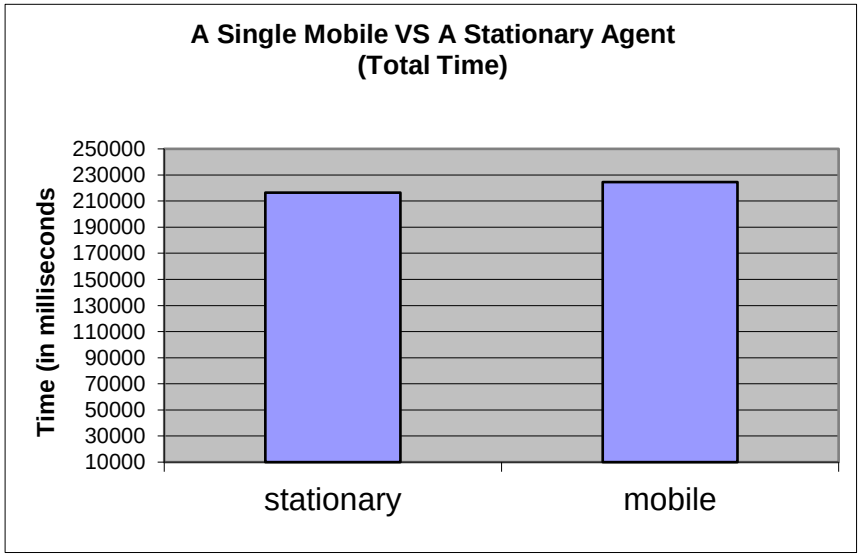

Figure 2: A single mobile agent VS A stationary agent (Total Time).

From figure_2 it could be seen that the stationary version was slightly swifter than the Mobile one in executing the tasks. This apparent difference may disappear in real-life applications when a steady state is reached. Table_ 3 shows that the stationary agent was nimbler than the Mobile agent in operations one and seven where both of them are matrix multiplication but the mobile agent was nimbler in operations four and ten both of them are matrix multiplications as well .in all remote document retrieval operations (two, five, eight) the Stationary version was a bit nimbler but this may not sustain when a steady state is reached. For archive update operations (three, six, nine) the stationary version was nimbler in operations three and six but it wasn't in operation nine.

In effect, this category can be concluded by saying that the execution time by both the stationary agent and the mobile agent in applications like the ones that have been presented here is more or less the same when a steady state is reached.

Multiple Mobile agents VS A Stationary agent.

In this category, there were multiple Mobile agents and a Stationary agent.

First, mobile agents without source code were dispatched to the remote host with requests for services from the stationary agent on the remote host. Upon almost concurrent arrival the Mobile agents submitted the requests to the stationary agent. The stationary agent accessed the data sources and

carried out the requested services (these were retrieving elements of the matrices and multiplying them, reading the documents and updating the archive) then it gave back the final results to the mobile agents. Mobile agents in this case neither bore source codes nor had access to the data source on the remote host. The Stationary agent carried the tasks in sequential manner and mobile agents had to wait for each other.

Second, multiple mobile agents with source code were dispatched to the remote host. Upon almost concurrent arrival they accessed the data sources and carried out the execution. The Stationary agent in this case neither bore source code nor accessed the data sources on the remote host. Mobile agents operate simultaneously and no shared data sources that could impose waiting time.

The main purpose here is to see which mode was faster in executing the operations, is that when the execution was carried out sequentially by the Stationary agent or when it being carried simultaneously by the visitors. That was determined by the time measured for each mode.

- **Code on A stationary agent.**

In this case, the assumption was that the mobile agents had concurrently arrived at the remote host each one had brought a request in the same order presented in Table_ 2. So, the stationary agent carried the requests sequentially as appears in Table_2. Agent one that carried matrix multiplication1 didn't have to wait and it would get its service in real time but the rest of the mobile agents did have to wait. When the stationary agent finishes the first service it will carry out the second one. So, the time required to accomplish the second service would be the time required to accomplish the first one and the real time needed to accomplish the second one from mobile two's perspective. For the second agent that carried operation two (remote document retrieval) the de facto time was operation one real time and operation two real time. Operation one time for the second mobile agent was the waiting time for that agent. The third mobile agent had to wait for the first mobile and second mobile. So, the third agent underwent waiting time which was the time required to execute operations one and two. This waiting time propagated linearly throughout the set of mobile agents waiting for accommodation

From Table 3, Table 4 has been constructed as the following:

Table 4: De facto Time

Operations Sequence	De facto Time
Operation 1	52262
Operation 2	66000
Operation 3	108231
Operation 4	132207
Operation 5	142421
Operation 6	173764
Operation 7	182414
Operation 8	190131
Operation 9	211536
Operation 10	216573

Table 4 presents the de facto time that includes the real-time and waiting time. From table 3 (real time), the pure real time required to accomplish each operation by the stationary version could be obtained. So, if the stationary agent was able to carry out the operations in parallel mode it might be able to get the real time for each operation. However, the stationary agent lacks this ability.

The purpose here is to compare the real time presented in Table 3 to the de facto time presented in table 4.

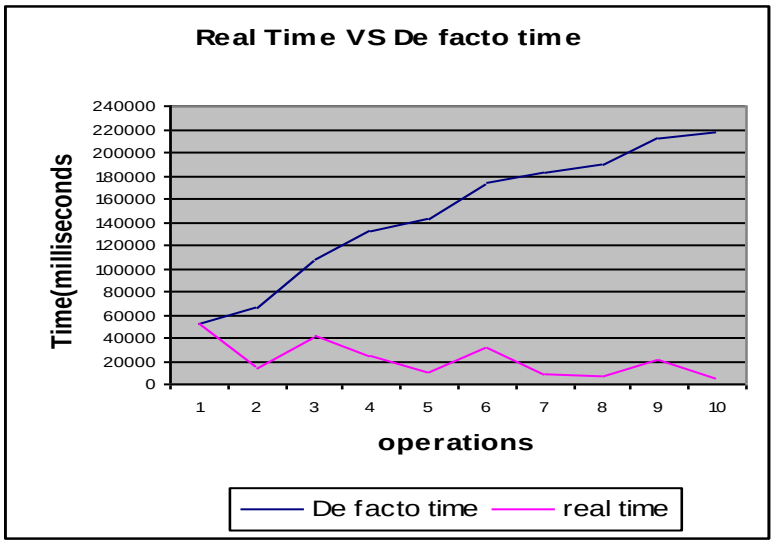

Figure 3: Real Time VS De facto Time**.**

Figure 3 presents how affected the mobile agents with the waiting time. For example, agent ten that carried operation ten had to wait for a time that required for accomplishing all operations from one through nine spite its real time was the shortest one.

- **Code on multiple Mobile agents.**

In this case, the assumption was that the mobile agents had concurrently arrived at the remote host each one had brought a request in the same order presented in Table 3. The source code had been carried along by the mobile agents. There was no dependency between agents in accessing the data sources. In other words, each mobile agent had its own data source on that remote host. So, a mobile agent didn't have to wait for anything. No waiting time at all. Mobile agents were operating simultaneously on the remote host.

After a series of trials on operations that are presented in Table 3 the following results have been obtained

Table 5: Competitiveness Time

Operations Sequence	Competitiveness Time
Operation 1	204378
Operation 2	82926
Operation 3	208234
Operation 4	141388
Operation 5	72404
Operation 6	187260
Operation 7	81200
Operation 8	64870
Operation 9	148058
Operation 10	50566

Table 5 reveals that despite being operating in parallel mode and with no dependencies between Mobile agents or waiting time, operations couldn't be accomplished in real time as appears in table 3. In effect, when multiple agents are operating on the same platform, they compete with each other on the

operating system resources. With Single agent operating on a platform all operating system resources are devoted to that single agent. But in the case of multiple agents operating on the same machine simultaneously, operating system resources are shared by all of them. the competition on the sources here affects the execution and accomplishing time. Time required to accomplish a task in this mode was denoted as competitiveness time. The purpose here was to compare the real time presented in Table_3 to competitiveness time presented in table_5.

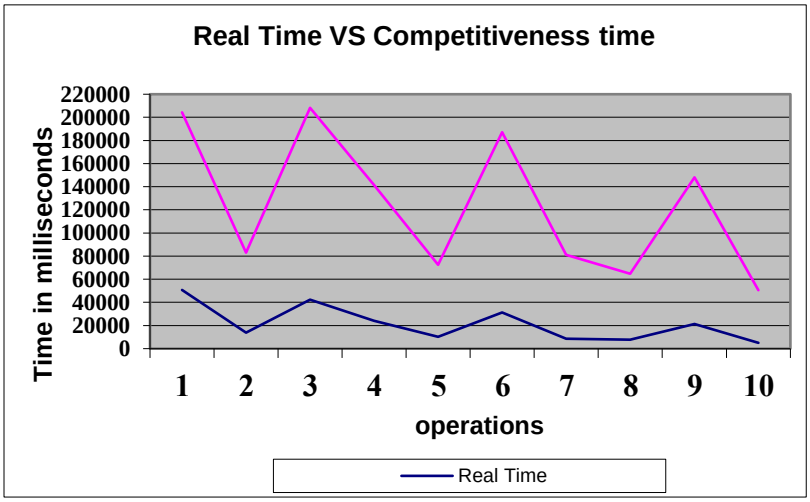

Figure 4: Real Time VS Competitiveness Time.

Figure_4 shows that the competitiveness time doesn't grow in a linear manner as waiting time. In effect, competitiveness time depends highly on real time. A proportion can be set between the real time and competitiveness time. As high the real time as high the competitiveness time will be, where waiting (de facto) time grows linearly without any relation to the real time.

Figure 5 depicts the relationship between Real Time, Competitiveness Time and Waiting Time.

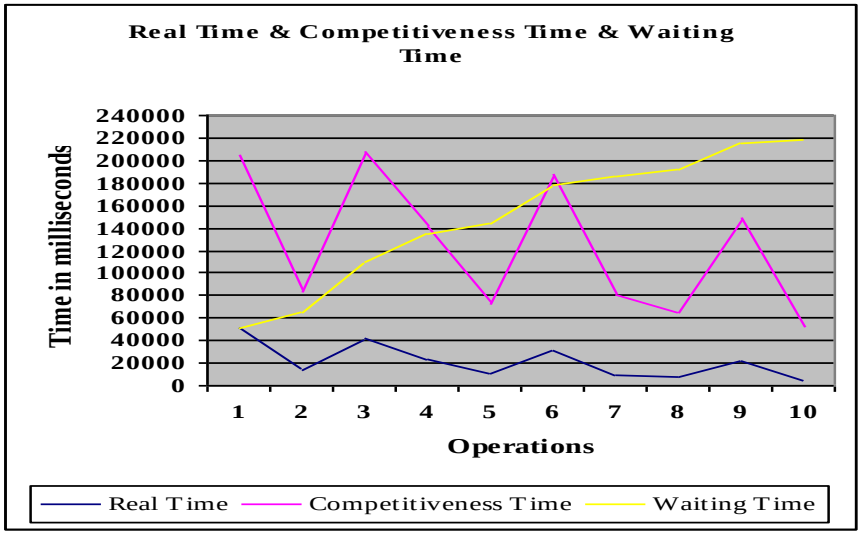

Figure 5: Real time & Competitiveness Time& Waiting Time.

This category could be concluded by saying that when the source code is held by the stationary agent, the visitors have to queue and wait for each other. Mobile agents with long-serving time operations and others with short-serving time operations are treated alike. With reference to table 3 Mobile agent that carries operation ten has to wait until the nine previous agents are served. Despite being the shortest serving time operation, Agent Ten undergoes waiting time which is the time required to accomplish all previous nine operations. In effect, this mode may be undesirable in real life applications where there is an infested platform. On the other hand, when the source code is held by the Mobile agents, they still

cannot accomplish their tasks in real time presented in table 3 that results from the competition between agents on the operating system resources, but with this mode agents do not have to wait for each other rather they compete each other and agents with short execution time operations can accomplish their tasks earlier than those with long execution time operations.

Future work

For security reasons, it's very desirable that the visiting mobile agents being kept away from accessing the data sources on the remote host. So, with this desire all operations on the remote host should be carried out by the stationary agent and the visiting agents being accommodated by the stationary agent

With reference to the methodology and implementation of this study it was clear that:

When there was a single mobile agent being accommodated by a stationary agent there was no difference in the performance between both of them and this desire could be easily fulfilled

The other category is when there are multiple agents operating concurrently on the same host. It was shown that when the execution being carried out by the stationary agent mobile agents have to wait for a long time before being accommodated because the stationary agent cannot operate in parallel it has to accomplish the requested tasks in a sequential manner (first-requested first-served). It was also seen that when the execution was being conducted by the visiting agents there was a competition on the operating system that has some impact on the performance.

With reference to figure_5 it could be seen that the competitiveness time is more acceptable than the waiting time. But once again no security can be guaranteed with multiple agents accessing the data sources on the remote host. So, the idea that when the source code is placed on the stationary agent the sluggishness of the execution, which results from the waiting time, is highly affects the performance, on the other hand when the source code is imbued into the mobile agents and those mobile agents have access to the data sources on the remote host security not guaranteed any more. Then if the problem of waiting time could be solved and the data sources be maintained on the remote host to be accessed only by the host side then the local sources may be saved from being corrupted by malicious mobile agents, meanwhile, the desired performance is still maintained.

The proposal here is that a master stationary agent should be maintained that receives requests from incoming visitors. When the master stationary is willing to serve a visitor, the Worker agent will be spawned by the master stationary agent to carry out that particular request for that particular visitor.

So, First, the visiting agents send their requests to the Master stationary agent. Once the master stationary agrees to accommodate the visiting agent, Worker agent will be spawned by the stationary agent. The Worker will inherit all necessary information from the master stationary including the identity of the visiting agent. Once the Worker finishes the process it forwards the results to the visiting agent and disposes itself.

Once this proposal is set then the results of its execution would be compared to that one presented in Table 4 (code on multiple mobile agents). In other words, the comparison would be between multiple mobile agents with source code and multiple stationary agents with source code this assessment will be in terms of execution time.

Conclusion

There are several conclusions that can be derived from this study:

- The results have shown that with a single mobile agent to a stationary agent, the performance is almost the same in applications similar to those that have been studied here.
- Multiple mobile agents accommodated by a stationary agent undergo waiting time that grows in a linear manner without discrimination between small operations and big operations.
- The results have also revealed that when multiple agents operate on the same machine simultaneously, they compete with each other on the operating system resources. This competition thwarts mobile agents from having their tasks finished in real time.
- Moreover, the results revealed that where the waiting time grows linearly, competition time could vary in the real time. For this reason, competition time may be favored over waiting time in real life applications as it discriminates between small operations and big operations.
- The first perspective of continuation for this work is in the development and evaluation of spawning Stationary agents to overcome security concerns.

References

[1] M. Shaw, “Myths and mythconceptions: What does it mean to be a programming language, anyhow?,” Proc. ACM Program. Lang., vol. 4, no. HOPL, 2022, doi: 10.1145/3480947.

[2] A. Mämmelä, J. Riekki, and M. Kiviranta, “Loose Coupling: An Invisible Thread in the History of Technology,” IEEE Access, vol. 11, no. June, pp. 59456–59482, 2023, doi: 10.1109/ACCESS.2023.3284685.

[3] Y. Fu, L. Liu, H. Wang, Y. Cheng, and S. Chen, “SFS: Smart OS Scheduling for Serverless Functions,” Int. Conf. High Perform. Comput. Networking, Storage Anal. SC, vol. 2022-November, pp. 1–15, 2022, doi: 10.1109/SC41404.2022.00047.

[4] D. G. Schwartz, Daniel G. Schwartz, “Review of ‘Knowing our world: an artificial intelligence perspective,’” Georg. F. Luger, Springer, pp. 1–3, 2023.

[5] “Tcl-Tk for EDA Tool,” in Programming and GUI Fundamentals, Wiley, 2022, pp. 185–210. doi: 10.1002/9781119837442.ch10.

[6] J. Barambones, J. Cano-Benito, I. Sánchez-Rivero, R. Imbert, and F. Richoux, “Multiagent Systems on Virtual Games: A Systematic Mapping Study,” IEEE Trans. Games, vol. 15, no. 2, pp. 134–147, Jun. 2023, doi: 10.1109/TG.2022.3214154.

[7] S. Kirrane, “Intelligent software web agents: A gap analysis,” J. Web Semant., vol. 71, p. 100659, Nov. 2021, doi: 10.1016/j.websem.2021.100659.

[8] P. K. Shukla, A. Aljaedi, P. K. Pareek, A. R. Alharbi, and S. S. Jamal, “AES Based White Box Cryptography in Digital Signature Verification,” Sensors, vol. 22, no. 23, 2022, doi: 10.3390/s22239444.

[9] A. Shashaj, F. Mastrorilli, M. Stingo, and M. Polito, “An industrial Multi-Agent System (MAS) platform,” pp. 1–12.

[10] M. Nieke, L. Almstedt, and R. Kapitza, “Edgedancer: Secure Mobile WebAssembly Services on the Edge,” in Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking, New York, NY, USA: ACM, Apr. 2021, pp. 13–18. doi: 10.1145/3434770.3459731.

[11] J. Cao, G. H. Chan, W. Jia, and T. S. Dillon, “Checkpointing and rollback of wide-area distributed applications using mobile agents,” Proc. - 15th Int. Parallel Distrib. Process. Symp. IPDPS 2001, vol. 00, no. C, pp. 1–6, 2001, doi: 10.1109/IPDPS.2001.924943.