British Journal of Mathematics & Computer Science
16(6): 1-8, 2016, Article no.BJMCS.26024
ISSN: 2231-0851

SCIENCEDOMAIN *international*
www.sciencedomain.org

Development of an Optimized Mobile Agent Migration Pattern for Pull-All Data Strategy

M. O. Oyediran[1*], T. M. Fagbola[2], S. O. Olabiyisi[1] and E. O. Omidiora[1]

[1]Department of Computer Science and Engineering, LAUTECH, Ogbomosho, Nigeria.
[2]Department of Computer Science, FUOYE, Oye – Ekiti, Nigeria.

Authors' contributions

This work was carried out in collaboration between all authors. Authors MOO and TMF both designed the study and carried out software implementation of the algorithm as well as statistical analysis. Authors SOO and EOO were supervised entire process of the design and implementation of the study. Author MOO wrote the first draft of the manuscript while authors TMF and SOO managed the literature searches and edited the manuscript. All authors read and approved the final manuscript.

Article Information

DOI: 10.9734/BJMCS/2016/26024
Editor(s):
(1) Qiang Duan, Information Sciences & Technology Department, The Pennsylvania State University, USA.
Reviewers:
(1) Kalpana L. Chaudhari, Shah and Anchor Kutchhi Polytechnic, Mumbai, India.
(2) Samer Ibrahim Mohamed, MSA University, Egypt.
(3) Ahmet Sayar, Kocaeli University, Turkey.
Complete Peer review History: http://sciencedomain.org/review-history/14728

Original Research Article

Received: 30th March 2016
Accepted: 28th April 2016
Published: 22nd May 2016

Abstract

A mobile agent is a software entity that migrates from one host to another, performing data collection and software configuration tasks on behalf of users in distributed networks. However, studies have shown that the existing pull – all data strategy of mobile agent migration pattern moves data from one host to the next host along with the mobile agent, thereby resulting in data accumulation which lowers the performance of the mobile agent in terms of transmission time and network load. This research developed an optimized mobile agent migration pattern for the pull-all data strategy in distributed networks using path strategy of Ant Colony Optimization (ACO) technique.
The obtained results shows that the optimized pull-all data migration pattern developed produced lower network load and transmission time than the existing pull-all data migration pattern.

Keywords: Mobile agent; migration pattern; pull-all; transmission time and network load.

**Corresponding author: E-mail: mooyediran75@lautech.edu.ng;*

1 Introduction

Mobile agents are autonomous active objects or software entities, migrating between different network locations, executing tasks locally and continuing their execution at the point where they stopped before migration [1]. One of the most attractive applications for mobile agents is the notion of "distributed information processing". This is particularly clear in the mobile computing scenarios where users have portable computing devices with only intermittent, low bandwidth connections to the main network. A mobile agent can abandon the portable device, move onto the network locations of the needed information resource and perform a locally custom retrieval task. Only the results are transmitted back to a portable device [2]. However, the wide spread of Java-based applications has made mobile agents to become extensively popular not only in the research community but also in industrial projects. Mobile agents have the following properties which distinguish them from other programs:

1. **Adaptability -** Mobility of agent required to learn about user's behavior and adapt it to suit the user. Indeed, to evolve adequately the differences between heterogeneous systems, the agent must be able to adopt the changes during the execution.
2. **Autonomy-** Mobile agent must be able to make his own decision to be performed to achieve the user's tasks, also he must be able to migrate from one machine to another in the network and execute the user's tasks.
3. **Communication -** Mobile agent must have the ability to communicate with others agents of the system in order to exchange information and benefit from the knowledge and expertise of other agents.
4. **Mobility-** Mobile agent has the ability to move from one host to another, either by moving the agent's code or by serializing both code and state to allow the agent to continue the execution in a new context.
5. **Persistence -** A persistent agent it will be able to retain the knowledge and state over extended period of time to be accessed later on. Once the mobile agent is set up, it is not dependent on system that has been initiated and it is automatically recovered when the agent is terminated or when it is flushed from memory to the database.

Mobile-agent systems differ from process-migration systems in that the agents move when they choose, typically through a ‘jump’ or ‘go’ statement, whereas in a process-migration system the system decides when and where to move the running process (typically to balance CPU load) [3]. Actually, there are two ways to move a mobile agent. The first way is to use the go command to initiate migration with a default migration behavior to a single remote agency, and the other way is to use so-called migration properties to configure the migration process in detail [4]. The existing pull-all migration pattern moves the data along with the mobile agent [5], as the mobile agent migrates from the home host to the next host, it moves with the data collected on the host and carries it to the next host on the itinerary, this method is repeated until all hosts on the itinerary are visited and returns all the collected data to the home host.

This is a challenging problem called data accumulation especially when the list of hosts to visit is large. This drawback lowers the agent’s performance in terms of transmission time and network load. This study focuses on developing an improved mobile agent migration pattern for the pull-all data strategy to realize more optimal and efficient migration process in distributed networks.

2 Related Works

Braun [4] developed a new migration model named Kalong, the developed model entails a detailed analysis of network load of mobile agents as compared to client-server approaches in several typical application scenarios shows the potential benefits of mobile agents. Malgi [6] developed a migration model called the STRING which is an approach to capture and re-establish the execution state of mobile agents programmed in Java, without any modifications to the Java virtual machine.

Osunade [5] developed a modified migration pattern for the data collected on a host and compare the performance with the existing data migration pattern using the transmission time and network load as evaluation metrics, the designed pattern entails simulation of mathematical model to compare the transmission time and network load generated when two different data migration patterns are implemented in a Java based mobile agent system. The models were coded in Java programming language and simulated using data from a typical mobile agent and computer network implementation. Osunade and Atanda [7] developed mathematical model for two mobile agent data migration pattern process and was programmed in java, the developed model was compared with the existing pattern for situations when data is migrated and not migrated. Mohamed [8] developed migration pattern for embeddable mobile agent platform supporting runtime code mobility. The authors presented the design and the implementation of Mobile-C, an IEEE Foundation for Intelligent Physical Agents (FIPA) compliant agent platform for mobile C/C++ agents. Such compliance ensures the interoperability between a Mobile-C agent and other agents from heterogeneous FIPA compliant mobile agent platforms. Also, the Mobile-C library was designed to support synchronization in order to protect shared resources and provide a way of deterministically timing the execution of mobile agents and threads. The mechanisms of agent migration and their synchronization were combined.

Nimbalkar et al. [9] developed load balanced process migration of mobile agents in Linux environments. This was an attempt to gain better performance in executing the processes in distributed environments, which allows execution environments to move processes from processor to processor dynamically at any point in the life of the process. A migration algorithm was developed to migrate the processes dynamically to lightly loaded host using the CPU load as migration criteria. During the migration, the algorithm took care of identifying the live nodes and migrate the process to lightly loaded workstation. The centralized server maintained the status and load of each node in the NOW.

Agents were running independently and are solely responsible for deciding if process need to be migrated or not based on the load of hosts. Agent node was responsible for keeping track of load and if it exceeds the defined threshold, decision was made to migrate the process. In a case decided, it identified the lightly loaded host and transferred state of the process to the target hosts. On the target hosts, the state was collected and the process resumed the execution and sent the result back to the source machine which initiated the request to migrate the process. By dynamically moving processes throughout their lifetimes, the system could potentially adapt better to changes in the load that could not be foreseen at the start of the tasks. This dynamic adaptation led to a better system-wide utilization of available resources than static process.

Shrouf et al. [1] investigated mobile Agent Optimization Patterns. The work extended classification of mobile agent design patterns to involve optimization patterns. Two (2) optimization design patterns for mobile agents namely: V-Agent Optimization Pattern and P-Agent Optimization Pattern were proposed based on mathematical computing model which support reusability of designs in mobile computing area. Sample of four master mobile agents that created three slave mobile agents was used. Master agents were working on set of clients using Aglet alpha release 2.0.5, Tahiti working server and Java Execution Environment (JEE) platform. Slave agents were created by master agents that received multiple messages from master agents. Master agents sent 4000 messages. Consequently, slave agents recorded message response time in milliseconds. Finally, optimized computed time was computed using the two.

3 Methodology

The objectives of this research is realized using the evolutionary and unified modelling languages in software engineering. These are derived from the developed mathematical model which is used by the mobile agent as the migration pattern to determine the shortest migration path between the home host and destination host for the pull-all data strategy in distributed networks. The developed algorithm consist of starting instructions that determine sub-optimal routes from start host to end host, compute path distances for each of the connected host and calculate the possible total distance for each sub-optimal route. The starting instructions are followed by sequential instructions that allow only current data on a host to be migrated to the home host, thereby removing the data accumulation stage. The developed migration pattern is simulated in OMNeT++ integrated environment and the performance is evaluated by comparing it with the existing migration pattern using the network load and transmission time as metrics.

The stepwise procedures for this research is as follows:

Mathematical model – the mathematical model is developed by modifying the ant equation to find the shortest path for the mobile agent to choose the shortest path distance from the home host to the destination host.

Developed Algorithm – the ants algorithm is modified with path optimization strategy and integrated into the existing pull – all data strategy.

3.1 Mathematical modelling of the developed migration pattern for the pull-all data strategy

The developed approach integrates the path optimization strategy of ants into the existing migration pattern for the pull-all data strategy. The optimization within the framework of an itinerary is a NP-complete problem which entails finding the shortest path that visits each given host exactly once. The migration pattern was developed using mathematical expression, in equation 1 and 2 respectively, the following assumptions were made, there exists a set of n variables; where n is the number of hosts on an agents' itinerary. Then, the construction graph is fully connected and corresponds to the map of hosts, with each host corresponding to a node and possible edges being the connection between the hosts, initially, an ant starts in a random host. The next host j for an ant to choose (host j), given it is currently in host i and has already visited the hosts in partial solution s_p, is given by:

$$P(j|s_p, i) = \frac{\tau_{il}^{\alpha} * \eta(il)^{\beta}}{\sum_{I \in N\, s_p} \tau_{il}^{\alpha} * \eta(il)^{\beta}}, \forall j \in N(s_p) \tag{1}$$

with α and β being weight parameters that determine the importance of the pheromone and heuristic function respectively. The equation states that only hosts that have not yet been visited are considered, and constitute the neighborhood $N(s_p)$.

The heuristic function $\eta(ij)$ is defined by the inverse distance between hosts i and j: $1/d_{ij}$, hence, the closer the host, the more probable it is to be chosen by the ant.

A good initial pheromone value for all the edges should be slightly higher than the amount added at each iteration, estimated roughly by $T_{ij} = T_0 =$ number_of_ants/C^{NN} with C^{NN} as the length of a tour generated by a nearest neighbor heuristic. The quality of the solution s is defined by the evaluation function $f(s) = 1/g(s)$ and measures the inverse of the total length of the tour g(s).

After each ant has constructed its solution, pheromone values are updated according to:

$$T_{ij} = (1 - \rho) * T_{ij} + f(s) * C \tag{2}$$

where C is a constant.

This equation states that the shorter the path chosen by the ant, the more pheromone is added to each of the edges corresponding to the path and it's used to model the migration pattern. At the end of an iteration, on the basis of the quality of the solutions constructed by the ants, the pheromone values are modified in order to bias ants in future iterations to construct solutions similar to the best ones previously constructed.

Algorithm development and Simulation parameters.

The developed algorithm is as follows:

```
Set parameters,
Initializing pheromone trails
Loop/* at this level is called a iteration*/
Each is positioned on a starting node
   Loop/* at this level is called a step*/
   Each ant applies a state transition rule to incrementally build a solution
   And a local pheromone updating rule
Initialize agent's itinerary parameters
    // start _ node, end _ node, numbers _ of _ nodes to visit
       (Originating server),                 routing path = 0
Agent's Migration Paths Construction
    //set Agent to passive state
Initiate Pull-All, (Start node [server]) to the first node, obtain data and back to server
        (1)  Determine Sub-optimal routes from start_node to end_node (termination node)
             // for m numbers of nodes, $2^m$ numbers of sub-optimal routes will be obtained.
        (2)  Compute path cost for each of the connected node on the sub-optimal route.
        (3)  Calculate the possible total distance for each sub-optimal route
             // $2^m$ numbers of distances will be obtained
        (4)  Value ordering of distance
        (5)  Update Agent's KB with RP parameter in the new routing path
        (6)  Update Agent Start_node = current_node
                                   End_node =Server
Until End_condition.
```

The developed flowchart is as follows:

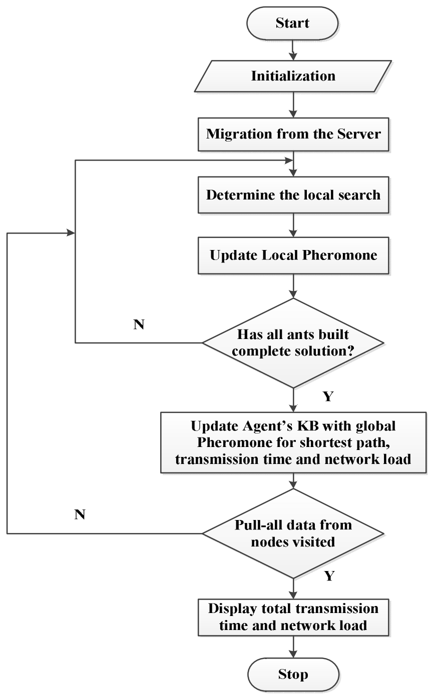
Start
Initialization
Migration from the Server
Determine the local search
Update Local Pheromone
Has all ants built complete solution?
N
Y
Update Agent's KB with global Pheromone for shortest path, transmission time and network load
Pull-all data from nodes visited
N
Y
Display total transmission time and network load
Stop

Fig. 1. Flowchart of the developed migration pattern for the pull-all data strategy

The system is set to active start and initializes the machine with the itinerary parameters i.e. the start node, end node, the total numbers of nodes to be visited, after which the agent migrate from the server to the next host. All possible paths from source to the destination for all the nodes to be visited is constructed and after which the local pheromone constructed is updated, if all the ants have built complete solutions, the global pheromone discovered (i.e. the shortest path) is updated in the knowledge base (KB) of the system and which triggers the system to start pulling data in all nodes visited using the global pheromone discovered, if not, the system determines the local search again. The nodes are checked to confirm if all nodes are visited which is to display the total transmission time and network load if all nodes are visited and if all nodes are not visited, the system is to determine the local search again.

4 Simulation and Result Analysis

In order to verify the performance of the migration pattern, simulation experiment was carried out in OMNeT++ integrated environment and executed using a combination of inputs. The inputs were varied to determine the impact of marshalling factor, number of hosts visited and amount of data migrated on the transmission time and network load for pull-all data strategy with the following parameter settings.

i. Network Area: 500 m x 500 m
ii. Transmission rate: 250 s
iii. Mobile agent code size: 70 packets/seconds
iv. Mobile agent data size: 5 packets/seconds
v. Mobile agent state: 3
vi. Number of host visited: 4
vii. Size and data on host: 5 packets
viii. Selective constant: 0.1 – 1.0
ix. Traffic Type: UDP

The interface is a GUI with two sections: one for data input and another for output, the input section provides a field for all the factors used in simulating the developed migration pattern for the pull-all data strategy. Thus, the input values could be varied to test various scenarios as applicable.

The result obtained from the simulation experiment is presented in Table 1 and the graph for transmission time across the itinerary is plotted against that of the existing and it's presented in Fig. 2 and the result for network load obtained from the simulation experiment is plotted against the existing and it's presented in Fig. 3.

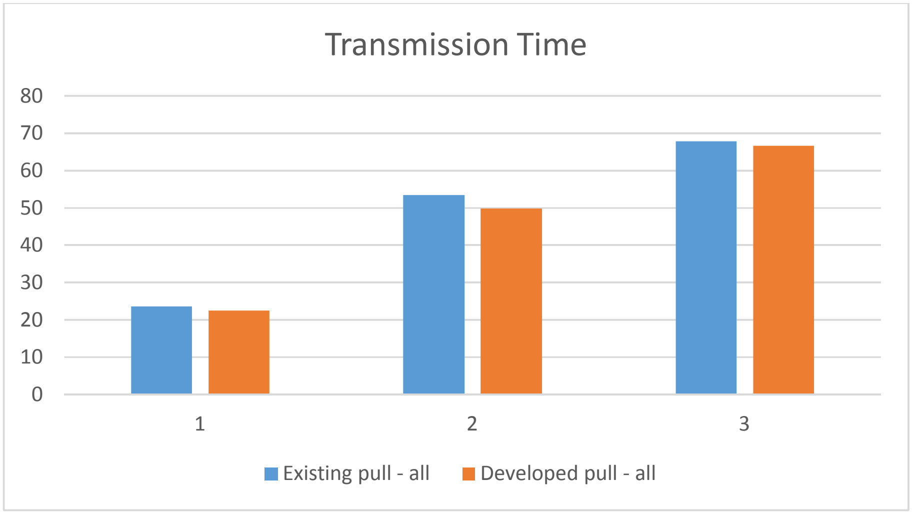

Fig. 2. Result for the transmission time

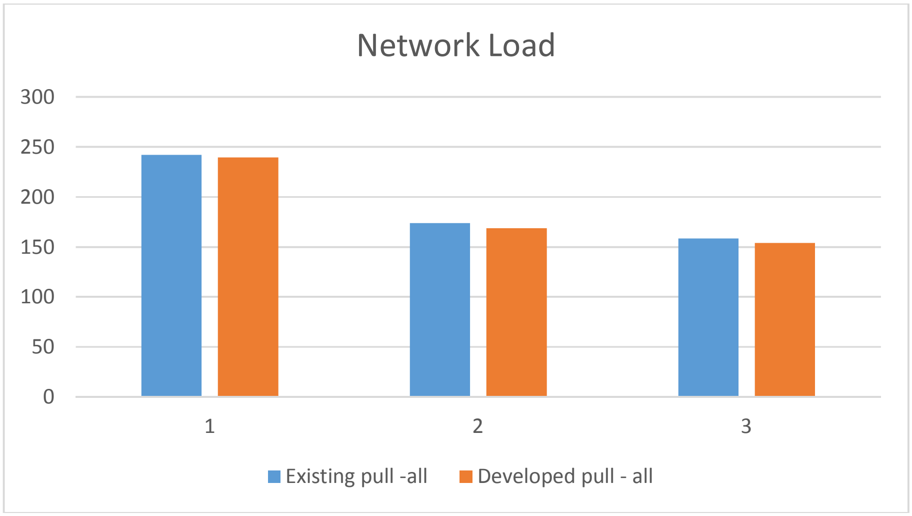

Fig. 3. Result for the network load

Table 1. Simulation result for the existing and developed migration pattern for pull-all data strategy

Nodes	Existing pull-all (Transmission time)	Developed pull-all (Transmission time)	Existing pull-all (Network load)	Developed pull-all (Network load)
Node 1	23.57	22.52	242.01	239.54
Node 2	53.42	49.79	173.71	168.96
Node 3	67.83	66.68	158.51	154.01
Total mean	144.52	138.99	574.21	559.51

The simulation results showed that the total mean transmission time for both the existing and optimized pull-all migration pattern were 144.52 s and 138.99 s, respectively, while the total mean network load obtained were 574.21 and 559.51, respectively and the chart is presented in Fig. 4.

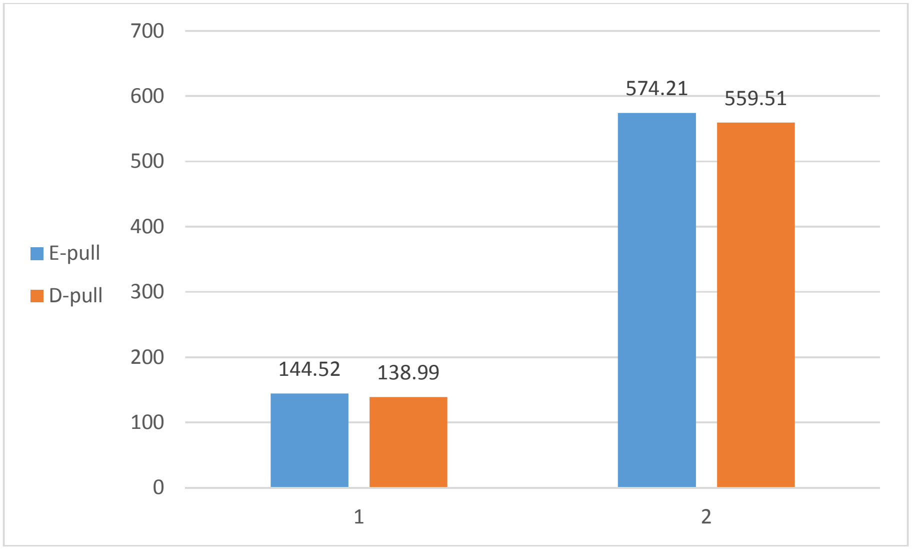

Fig. 4. Total mean of transmission time and network load

5 Conclusion

As lower network load and transmission time interpret into better performance of the migration pattern; the results obtained from all the evaluations conducted in this research work reveal that in general, the developed pull-all data migration pattern developed produced lower network load and transmission time than the existing pull-all data migration pattern. Consequentially, the modified data migration pattern performed optimally and more efficiently in a significant manner than the existing data migration pattern.

Competing Interests

Authors have declared that no competing interests exist.

References

[1] Shrouf FA, Turani A, Baker AA, Omri AA. Analysis of mobile agent optimization patterns. British Journal of Applied Science & Technology. 2014;4(12):1841-1857.

[2] Robert SG, David K, Ronald AP, Joyce B, Daria C, Peter G, Martin H. Mobile-agent versus client/server performance scalability in an information-retrieval task. Proceeding of Springer-verlag. 2004;1-16.

[3] Ojesanmi OA. Security issues in mobile agent applications. International Journal of Agent Technologies and Systems. 2010;2(4):39-55.

[4] Braun P. The migration process of mobile agents: Implementation, classification and optimization. PhD Thesis. Friedrich-Schiller-Universitat Jena. 2003;25-293.

[5] Osunade. Data migration patterns for java based mobile agent system. Unpublished Ph.D Thesis, in the Department of Computer Science, University of Ibadan. 2007;1-80.

[6] Malgi A, Bansod N, Choi BK. Efficient implementation of strongly migrating mobile agents in Java. Technical Report Dept. of Computer Science, Michigan Technological University, USA. 2005; 04(04):110-117.

[7] Osunade S, Atanda FA. Analysis of two mobile agent migration patterns. Journal of Mobile Communication. 2008;2(2):64-72.

[8] Mohamed B, Khaoula A, Noreddine G. Communication and migration of an embeddable mobile agent platform supporting runtime code mobility. International Journal of Advanced Computer Science and Applications. 2012;3(1):50-56.

[9] Nimbalkar MV, Nagargoje HM, Pathak GP, Vishnudas MB. Mobile agent: Load balanced process migration in Linux environments. Advances in Software Engineering and Systems. 2013;146-150.