Performance Evaluation of Remote Procedure Call and Mobile Agent for Network Monitoring

Arekete, S.A¹, Oguntunde, B.O²
¹,²Computer Science Department, Redeemer’s University Ede, NIGERIA

ABSTRACT

This study examines the performances of remote procedure call (RPC) and mobile agents. Under increasing number of nodes, the performances of the two paradigms were considered. The results obtained show that the performances of the two are similar with few nodes or RPC even performed better in some cases. However, mobile agents gain significant advantages over RPC in terms of load overhead and speed when the number of nodes is increased significantly. Apart from savings in terms of load overhead and speed, it has been established also that mobile agents operate well under low communication bandwidth.

Keywords-- RPC, Client/Server, Mobile agent

I. INTRODUCTION

In a distributed computer system, communication between entities can be supported by various kinds of paradigms depicted in Figure 1 [1]. Message passing is the most basic; this lets processes communicate by explicitly sending and receiving messages. In this regard, asynchronous and synchronous message passing have been advocated. Asynchronous type of message passing is very flexible and hence supports a great variety of communication patterns. Developing distributed applications based on message passing primitives is, however, a very complex and error-prone task as the programs are hard to analyse and debug.

The remote procedure call (RPC) offers a higher-level communication abstraction. In RPC, to communicate, processes call remote procedures rather than explicitly sending and receiving messages. RPC supports client/server style of interactions: clients issue requests to servers, which execute the requested procedure and then return the results. Many RPC mechanisms support both asynchronous and synchronous calls. With RPC, a lot of the complexity inherent to reliable communication is hidden in stub procedures, which are generated automatically from interface specifications. For example, stubs provide methods for marshalling/unmarshalling of messages, encoding/decoding in heterogeneous environments as well as failure recovery. For RPC to work correctly, the called procedure must be available on the respective remote node, a requirement which, however, limits the usability of the RPC concept in large open distributed systems. It is therefore more desirable in many cases to ‘ship’ a procedure to a remote node and execute it there. For example, a client might move an application-specific filter to a remote database to access and compress data locally, without involving the system administrator of the remote node. This level of flexibility is provided by remote evaluation, which is a natural extension of the RPC abstraction [1].
code transfer ‘push’ and ‘pull’ approaches are conceivable. For example, Java Applets or ActiveX Controls are technologies allowing clients to pull code from servers, while Java Servlets are code pushed to servers. All approaches that provide for pushing or pulling code (plus parameters) from one node to another are grouped under the concept of remote execution. Hence, remote evaluation is considered to be a special case of remote execution [1].

While remote execution only allows for code mobility, the concept of a mobile agent supports process mobility, in which program executions may migrate from node to node in a computer network [2]. For migrating agents not only code but also the state information of the agent has to be transferred to the destination. An agent’s state is subdivided into data state (made up of agent’s global variables and instance variables) and execution state (comprising the local variables and the active threads). According to [3], mobile agent is an active entity which can migrate from one place to another one to meet the other agents and to access the services available in other nodes.

Two types of migration are distinguished in literature, namely: weak and strong migration. In strong migration, the underlying system captures the entire agent state consisting of data and execution state and transfers it together with the code to the next location [1]. Once at its new location, the agent’s state is automatically restored. From a programmer’s perspective, this scheme is very attractive since capturing, transfer and restoration of the complete agent state is done transparently by the underlying system. However, providing this degree of transparency in heterogeneous environments requires a global model of agent state and a transfer syntax for this information. Moreover, an agent system must provide functions to externalise and internalise agent state; only few languages allow this at such a high level, e.g., Facile or Tycoon. Moreover, complete agent state can be large and for multi-threaded agents, strong migration might be a very time-consuming and expensive operation.

To ameliorate the demands of strong migration, the development of the weak migration scheme has been made, where only data state information is transferred [1]. The size of the transferred state information can be limited even more by letting the programmer select the variables making up the agent state; he encodes the agent’s relevant execution states in the program variables. The programmer, can in addition, specify the sort of a start method that decides, on the basis of the encoded state information, where to continue execution after migration. While this method may substantially reduce the amount of state to be communicated, it puts additional burden on the programmer and makes agent programs more complex.

On the surface, strong agent migration and the concept of process migration are quite similar, but there is a major difference between them. With process migration, the decision to move a process to some node at what time is made by the underlying system, hence, from the applications’ point of view process migration is totally transparent. In contrast, agent migration is totally under control of the agent program itself. The agent decides when to move to which place depending on its own state. Consequently, agent languages need to provide specific statements, like a “Go-statement”, to let the agent initiate migration.

Single mobile agent systems and multiple mobile agents have been used in various systems. When a single agent is used, it can migrate through the entire network, moving from node to node, performing functions for the owner. However, when multiple agents are used, each mobile agent covers only a segment of the entire network. This can lead to faster processing, though more complex coordination would be required.

II. LITERATURE REVIEW

[4] Remark that RPC has become a popular paradigm for distributed application development in a client-server environment owing to the fact that it is simple, flexible and powerful. However, most of the existing RPC mechanisms are synchronous in nature, and hence fail to exploit fully the parallelism inherent in distributed applications. The authors therefore proposed a transport independent asynchronous RPC mechanism code-named ASTRA that combines the advantages of both RPC and the message-passing. “ASTRA calls do not block the caller (client) and the replies can be received as and when they are needed, thus allowing the client execution to proceed locally in parallel with the server invocation. All the calls are received and executed by the server in the order called by the client. ASTRA is unique among other asynchronous RPC systems in allowing its users to explicitly specify whether low-latency or high-throughput is required for a call” [4]. Load balancing in distributed environment in which RPC is used for communication has been proposed in [5]. The authors argued that while distributed environment has primary advantages of high performance, availability, and extensibility at low cost, when load of running applications are more and limited number of nodes present, then sharing of applications would definitely improve the performance of system. On the other hand, [6] stressed that mobile agent technology is a more suitable paradigm to facilitate distributed computing environment based applications than the conventional approaches like client server model and RPC; data gathered by the mobile agent from all the preceding hosts is carried to all the succeeding hosts, and after visiting as many hosts, the mobile agent returns to its owner with the data congregated from various remote hosts. However, as mobile agents can have attendant security issues in moving from host to host, the study suggests that before leaving a remote host, mobile agent should send the data gathered to the owner before leaving the remote host. The popularity of mobile agents over RPC and other traditional systems have been underscored in its usage in various applications including e-commerce, information processing.
distributed network management, and database access. “Information search and retrieval can be conducted by mobile agents in a decentralized system. As compared with the client/server model, the mobile agent approach has an advantage of saving network bandwidth and offering flexibility in information search and retrieval” [7]. The study presents a model for mobile agents to select the most reputable information host to search and retrieve information.

A performance comparison of remote procedure calling and mobile agent approach to control and data transfer in distributed computing environment is presented in [8]. The conceptual design of the MA is process-based and packet switching oriented. The mobility infrastructure is developed to facilitate a transmission control protocol/Internet protocol socket-based connection between source and destination machine using agent transfer. To specify the path of itinerary agent, a model of dynamic route decisions is defined and implemented using concept of oldness vector and random number generator. The proof of superior performance of MA to the traditional scheme is shown by developing an analytical model. Bandwidth usage against the number of requests per service, percentage denial of service measured against number of requests per service, percentage denial of service measured against failure rate, and network bandwidth overload with retransmission were parameters determined through the analytic model. The simulated network has a state that is modelled after Bernoulli random variable (BRV) with probability of success \( p = 80\% \) and probability of failure \( q = 20\% \). The network breakdown was implemented by an event generated by a multiplicative congruential pseudorandom number generator. The computer programs for the mobile agent and the simulated model are developed in C++ and Java programming languages and experimental results based on the simulation is presented. The results of the simulation shows that when the number of requests per service increases, the bandwidth usage against the total time for the network connection in RPC based scheme increases proportionately, thereby making it more liable to failure. At a fixed failure rate of 20\%, the mobile agent was found to perform better as the number of requests per service increases while RPC scheme deteriorates. When the level of failure rate is varied while measuring the percentage denial of service for the two different schemes, and the message size is fixed at 2 bytes, there is higher level of fault tolerance for MA. RPC fails totally at a network failure rate of 30\%. It is concluded that the MA based approach to control and data transfer is found to provide a more superior, efficient, dynamic, autonomous scheme, with a high level of modularity, flexibility, decentralized control, and asynchronous suitable for complex system than the existing technique (RPC).

Before entrusting tasks to a mobile agent, the agent owner stores in the mobile agent, a private key and the coding of the mobile agent. Using mobile agents offers important alternative to conventional approaches, such as remote procedure calls (RPCs) for Internet and mobile computing applications [9].

Extensive discussions on mobile agent mechanisms and applications have been discussed in [10], [11] and [12]. A comparative analysis of single mobile agent and multiple mobile agent systems have been examined in [2].

### III. METHODOLOGY AND PERFORMANCE COMPARISON

The performance of a mobile agent is benchmarked against the remote procedure call (RPC), which is a client-server paradigm in this study. This evaluation is aimed at gaining insights into the possible gains of mobile agent over RPC, or vice versa.

First, the network load of RPC is compared with that of MA. In RPC, the source node serves as a client that request services from each of the target nodes (or servers) sequentially. The client sends request to the servers and a result is sent back to the client. On the other hand, a mobile agent is dispatched to the target node, where local processing and information retrieval is done, and the MA moves to the next node, until the last target node is visited and MA returns to the source node.

Let the size (in bytes) of RPC request be denoted by \( B_{req} \) bytes and the result from the server represented by \( B_{res} \) bytes. Then, the total network load for RPC for \( m \) servers is given by:

\[
B_{RPC} = m(B_{req} + B_{res}) \quad \cdots \cdots \cdots \cdots (1)
\]

It is noted that for each server that the client communicates with using the RPC, there is a two-way communication, that is, the client sends a request to the server and the server replies with a result. Hence, if there are \( m \) servers, there will be \( m \) requests and \( m \) results (Eq. 1).

![Figure 2 Remote Procedure Call versus Mobile Agent](image)

With the MA approach, the agent migrates from its source node to target node \( T_1 \) incurring cost...
implications for code, state, and data transmission. This is carried to the next node where further data is accumulated, until all nodes are visited and MA returns to the source node. Therefore, the network load for the mobile agent approach is derived as follows:

The load $B_h$ of MA from home to the first target node is calculated as:

$$B_f = d_h + \sum_{k=1}^{m} d_k + B_s$$

where $d_k$ is the incremental load gathered from each target node. When the agent migrates back to its home, the final load is given by:

$$B_f = d_h + \sum_{k=1}^{m} d_k + B_s$$

Hence, network load of the mobile agent in roundtrip is given as:

$$B_{MA} = B_h + B_m + B_f$$

According to [13], typical sizes (in bytes) of mobile agent codes, state, and data, and RPC requests and results are given as follows:

- mobile agent code, $B_c = 3000$ bytes,
- mobile agent state, $B_s = 300$ bytes,
- RPC request, $B_{req} = 100$ bytes,
- reply, $B_{res} = 20000$ bytes,
- initial MA data, $d_h = 5000$ bytes,
- and incremental data at target nodes, $d_k = 5000$ bytes.

Using these values and plugging them into Equations 1 to 5, the performances of the RPC and MA are simulated. The relationship between number of target nodes and network loads due to RPC and MA are depicted in Figure 3. It is seen that when the number of target nodes are few, mobile agent exert more loads on the network. However, when the number of target nodes becomes large, RPC’s load grows faster than MA’s load. This is due to the fact that MA does processing at the target nodes and simply carries the summary results.

When the itinerary of MA is partitioned into sub-itineraries and a mobile agent visits only a segment of the total target nodes, the load of the mobile agent is further reduced. In Figure 3, the load of MA when the itinerary is broken into four sub-itineraries with a mobile agent visiting each sub-itinerary is seen to have very low cost implication ($B_{ma_4}$).

To determine the roundtrip time of RPC and MA, the delay or latency of the network between pair of nodes and the network throughput between pair of nodes is taken into consideration. Let $\delta(T_i, T_j)$ represent the network delay (latency) between nodes $T_i$ and $T_j$ and $\tau(S_h, T_i)$ represent the throughput of the network between nodes $S_h$ and $T_i$; then the response time to RPC, based on equation (1) is given by:

$$T_{RPC} = \sum_{i=1}^{m} \delta(S_h, T_i) + \frac{B_{req} + B_{res}}{\tau(S_h, T_i)}$$

It is assumed that the source node is $S_h$. The response time for a simple RPC is the sum of the time for transferring the request and result plus the delay for this network connection. The response time for MA,
based on equations (7 - 11). The execution time to migrate an agent from the home platform is given as:

\[ T_h = 2\mu(B_d + B_s) + \delta(S_h, t_1) + \frac{B_h}{\tau(S_h, t_1)} \]  

(7)

Marshalling and unmarshalling data and state information each takes \( \mu(B_d + B_s) \) unit of time. We define the length of data and state information of mobile agent

\[ B_{d,s} = d_h + \sum_{k=1}^{m} d_k + B_s \]  

(8)

Time agent migrates from first to last target platforms is:

\[ T_m = 2\mu(B_{d,s}) + \delta(t_k, t_{k+1}) + \frac{B_m}{\tau(t_k, t_{k+1})} \]  

(9)

In addition, the time to migrate from last target node back to the home node is given by:

\[ T_f = 2\mu(B_f(T,S)) + \delta(T_m, S_h) + \frac{B_f}{\tau(T_m, S_h)} \]  

(10)

Hence, total execution time is:

\[ T_{total} = T_h + T_m + T_f \]  

(11)

The execution time to migrate an agent from the home platform is given as:

\[ T_h = 2\mu(B_d + B_s) + \delta(S_h, t_1) + \frac{B_h}{\tau(S_h, t_1)} \]  

(12)

Marshalling and unmarshalling data and state information each takes \( \mu(B_d + B_s) \) unit of time. We define the length of data and state information of mobile agent

\[ B_{d,s} = d_h + \sum_{k=1}^{m} d_k + B_s \]  

(13)

Time agent migrates from first to last target platforms is:

\[ T_m = 2\mu(B_{d,s}) + \delta(t_k, t_{k+1}) + \frac{B_m}{\tau(t_k, t_{k+1})} \]  

(14)

In addition, the time to migrate from last target node back to the home node is given by:

\[ T_f = 2\mu(B_f(T,S)) + \delta(T_m, S_h) + \frac{B_f}{\tau(T_m, S_h)} \]  

(15)

Hence, total execution time is:

\[ T_{MA} = T_h + T_m + T_f \]  

(16)

According to Braun and Rossak (2005), typical values of the delay between source node and target node, \( \delta = 90 \text{ msec} \), throughput, \( \tau = 100 \text{ Kbytes/sec.} \), and time to marshall and unmarshall the mobile agent for each pair of nodes, \( \mu = 1 \text{ msec} \). Plugging these values into Equations 12 to 16, a simulation of the turnaround time based on RPC and MA versus number of target nodes in the results graphically depicted in Figure 4. It can be seen from Figure 4 that when the nodes are very few, RPC has lower turnaround time than MA. However, after the changeover point (around node 8), RPC has higher roundtrip time. In other words, MA becomes faster than RPC in execution. When the itinerary is partitioned into four and each sub-itinerary is visited by a mobile agent, the turnaround time becomes significantly lower (\( T_{ma,4} \)).

\[ B_{d,s} = d_h + \sum_{k=1}^{m} d_k + B_s \]  

(13)

Time agent migrates from first to last target platforms is:

\[ T_m = 2\mu(B_{d,s}) + \delta(t_k, t_{k+1}) + \frac{B_m}{\tau(t_k, t_{k+1})} \]  

(14)

In addition, the time to migrate from last target node back to the home node is given by:

\[ T_f = 2\mu(B_f(T,S)) + \delta(T_m, S_h) + \frac{B_f}{\tau(T_m, S_h)} \]  

(15)

Hence, total execution time is:

\[ T_{MA} = T_h + T_m + T_f \]  

(16)

According to Braun and Rossak (2005), typical values of the delay between source node and target node, \( \delta = 90 \text{ msec} \), throughput, \( \tau = 100 \text{ Kbytes/sec.} \), and time to marshall and unmarshall the mobile agent for each pair of nodes, \( \mu = 1 \text{ msec} \). Plugging these values into Equations 12 to 16, a simulation of the turnaround time based on RPC and MA versus number of target nodes in the results graphically depicted in Figure 4. It can be seen from Figure 4 that when the nodes are very few, RPC has lower turnaround time than MA. However, after the changeover point (around node 8), RPC has higher roundtrip time. In other words, MA becomes faster than RPC in execution. When the itinerary is partitioned into four and each sub-itinerary is visited by a mobile agent, the turnaround time becomes significantly lower (\( T_{ma,4} \)).

\[ B_{d,s} = d_h + \sum_{k=1}^{m} d_k + B_s \]  

(13)

Time agent migrates from first to last target platforms is:

\[ T_m = 2\mu(B_{d,s}) + \delta(t_k, t_{k+1}) + \frac{B_m}{\tau(t_k, t_{k+1})} \]  

(14)

In addition, the time to migrate from last target node back to the home node is given by:

\[ T_f = 2\mu(B_f(T,S)) + \delta(T_m, S_h) + \frac{B_f}{\tau(T_m, S_h)} \]  

(15)

Hence, total execution time is:

\[ T_{MA} = T_h + T_m + T_f \]  

(16)

According to Braun and Rossak (2005), typical values of the delay between source node and target node, \( \delta = 90 \text{ msec} \), throughput, \( \tau = 100 \text{ Kbytes/sec.} \), and time to marshall and unmarshall the mobile agent for each pair of nodes, \( \mu = 1 \text{ msec} \). Plugging these values into Equations 12 to 16, a simulation of the turnaround time based on RPC and MA versus number of target nodes in the results graphically depicted in Figure 4. It can be seen from Figure 4 that when the nodes are very few, RPC has lower turnaround time than MA. However, after the changeover point (around node 8), RPC has higher roundtrip time. In other words, MA becomes faster than RPC in execution. When the itinerary is partitioned into four and each sub-itinerary is visited by a mobile agent, the turnaround time becomes significantly lower (\( T_{ma,4} \)).

\[ B_{d,s} = d_h + \sum_{k=1}^{m} d_k + B_s \]  

(13)

Time agent migrates from first to last target platforms is:

\[ T_m = 2\mu(B_{d,s}) + \delta(t_k, t_{k+1}) + \frac{B_m}{\tau(t_k, t_{k+1})} \]  

(14)

In addition, the time to migrate from last target node back to the home node is given by:

\[ T_f = 2\mu(B_f(T,S)) + \delta(T_m, S_h) + \frac{B_f}{\tau(T_m, S_h)} \]  

(15)

Hence, total execution time is:

\[ T_{MA} = T_h + T_m + T_f \]  

(16)

According to Braun and Rossak (2005), typical values of the delay between source node and target node, \( \delta = 90 \text{ msec} \), throughput, \( \tau = 100 \text{ Kbytes/sec.} \), and time to marshall and unmarshall the mobile agent for each pair of nodes, \( \mu = 1 \text{ msec} \). Plugging these values into Equations 12 to 16, a simulation of the turnaround time based on RPC and MA versus number of target nodes in the results graphically depicted in Figure 4. It can be seen from Figure 4 that when the nodes are very few, RPC has lower turnaround time than MA. However, after the changeover point (around node 8), RPC has higher roundtrip time. In other words, MA becomes faster than RPC in execution. When the itinerary is partitioned into four and each sub-itinerary is visited by a mobile agent, the turnaround time becomes significantly lower (\( T_{ma,4} \)).

![Figure 4 Rountrip time of RPC and MA](image-url)

**IV. CONCLUSION**

From the above simulation, it can be concluded that when the number of nodes is few, RPC is a better choice for distributed processing. However, when we are dealing with large number of target nodes, MA is more advantageous for distributed processing and network monitoring as it consumes less network resources and has faster turnaround time. Furthermore, the fact that multiple copies of MA can be used would even increase...
the gains of MA the more. In addition to this, for RPC, there must be perfect connection between the client and the next target server to be queried. In MA, after the agent has taken off from the source node, the source node could be disconnected from the other systems until the agent is ready to migrate back to the source after its roundtrip.

REFERENCES


