Journal of Theoretical and Applied Information Technology
31st July 2017. Vol.95. No 14
© 2005 – ongoing JATIT & LLS

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

# A REVIEW OF FINDINGS ON AGENT SPAWNING AND MOBILITY

**OSAMA TREEF ALSHAKI, MOHD SHARIFUDDIN AHMAD, MOAMIN A. MAHMOUD**
College of Computer Science and Information Technology, Universiti Tenaga Nasional

E-mail: osamashaki@gmail.com, sharif@uniten.edu.my, moamin@uniten.edu.my

## ABSTRACT

Mobile agent technology is becoming more popular and has been implemented in many areas. Several research have been conducted to address its challenges including two of the most important which are agent spawning and agent mobility. This paper reviews the mobile agent technology, the background concept of mobile agent cloning and spawning, agent mobility as well as the problems faced by many researchers during their research on mobile agent. Various mobile agent types are also discussed. The paper finally proposes a new agent spawning and mobility models to resolve some of the researchers' problems.

**Keywords**: *Agent Architecture, Mobile Agent, Agent Cloning, Agent Spawning*

## 1. INTRODUCTION

Recently, the distributed agent concept has become a new computing paradigm in Internet distributed computing, including mobile computing. Mobile agent cloning and spawning are some of the most important techniques that are deployed for performing distributed tasks. The cloning technique may not be the best approach in real network environments mainly due to the fluctuation of network traffic, such as connection failures or heavy traffic on the network. For better performance, it is necessary that mobile agents be more sensitive to the network conditions.

We can classify agents, which are distributed over the network, into two types: static and mobile agents [1]. The static agent has the function of providing a mobile agent with node resources. Mobile agents are allowed to travel from one node to another. A mobile agent migrates to a node where services are being provided, and then returns to its starting point, namely the home node, after obtaining a service offered remotely [2]. One of the major potential application areas for mobile agents is distributed information retrieval, which involves access to a huge amount of data across a network [3, 4, 5]. In conventional distributed computing, the distributed information retrieval process is carried out through a direct connection mechanism, such as Remote Procedure Calls (RPC), which accesses the distributed database directly from a remote area.

In this paper, we review the mobile agents, agent types, agent cloning and spawning and propose models for agent spawning and mobility, an alternative approach to the problem of local agent overloads. Our paradigm entails that agents may spawn, pass tasks to others, migrate to another host, execute a specific task and die. The rest of the paper is organized as follows: Section 2 reviews the related work in software agent technology and mobile agent, and discusses some work on agent cloning and mobility. Section 3 presents a discussion of the review. Section 4 proposes a framework for agent spawning and mobility. Section 5 presents a framework for dynamic spawning of agents and Section 6 concludes the paper.

## 2. RELATED WORK

### 2.1 Agent Architecture

Researchers working in the area of agent architecture are concerned with the design and construction of agents that enjoy the properties of autonomy [6], reactivity, pro-activeness [7, 8], and social ability [9, 10, 11]. Wooldridge [12] states that agent architecture is essentially a map of the internals of an agent — its data structures, the operations that may be performed on these data structures, and the control flow between these data structures. Three classes of agent architectures can be identified [13]:

- Deliberative or symbolic architectures are those designed along the lines proposed by traditional symbolic AI.
- Reactive architectures are those that eschew a central symbolic representations of the agent's environment, and do not rely on symbolic reasoning, and

- Hybrid architectures are those that marry the deliberative and reactive approaches [9].

Wooldridge and Jennings [13] indicate that agent architectures can be viewed as software engineering models of agents and identify the above mentioned classes of agent architectures. Wooldridge [12] considers four classes of agents. Table 1 enumerates and gives a short description of each class. In our opinion, most agents follow one of the following four architectural classes.

*Table 1: Agent classes*

| Agent class | Description |
|---|---|
| 1. Logic based agents | In which decision making is realized through logical deduction (Wooldridge, 1999). |
| 2. Reactive agents | In which decision making is implemented in some form of direct mapping from situation to action (Wooldridge, 1999). |
| 3. Belief-desire-intention (BDI) agents | In which decision making depends upon the manipulation of data structures representing the beliefs, desires, and intentions of the agent (Wooldridge, 1999). |
| 4. Layered architectures | In which decision making is realized via various software layers, each of which is more-or-less explicitly reasoning about the environment at different levels of abstraction (Wooldridge, 1999). |

## 2.2 Agent Communication Languages (ACLs)

The difficulty to precisely handle coordination and communication increases with the size of the agent-based software to be developed. A number of languages for coordination and communication have been proposed [14]. Weíβ [15] enumerates a list of such languages. Table 4 describes the most prominent examples of agent communication languages (ACLs) according to Weíβ [15].

*Table 2: Most prominent agent communication languages*

| Agent communication language | Description |
|---|---|
| 1. KQML ("Knowledge Query and Manipulation Language") | It is perhaps the most widely used agent communication language [15]. |
| 2. ARCOL ("ARTIMIS Communication Language") | It is the communication language used in the ARTIMIS system [15]. ARCOL has a smaller set of communication primitives than KQML, but these can be composed [15]. |
| 3. FIPA-ACL (FIPA Agent Communication Language) | It is an agent communication language that is largely influenced by ARCOL [15]. Together FIPA-ACL, ARCOL, and KQML establish a quasi-standard for agent communication languages [15]. |
| 4. KIF ("Knowledge Interchange Format") | It is a logic-based language that has been designed to express any kind of knowledge and meta-knowledge [15]. KIF is a language for content communication, whereas languages like KQML, ARCOL, and FIPA-ACL are for intention communication [15]. |
| 5. COOL ("Domain independent coordination Language") | It aims at explicitly representing and applying coordination knowledge for multi-agent systems and focuses on rule-based conversation management [15]. Languages like COOL can be thought of as supporting a coordination/communication (or "protocol-sensitive") layer above intention communication [15]. |

### 2.3 Mobile Agent

Mobile agents are independent, smart programs that move through a network, seeking and interacting with various available/compatible services on a user's behalf. Mobile agent systems use specialized servers to interpret the agent's behavior and communicate with other servers on the network. They have inherent navigational autonomy and find their path through the network. Such agents can operate independently and perform tasks autonomously, if so desired. The runtime environment could be a closed-proprietary system or the open Java environment.

Mobile agents can be executed on all types of computers because their agent code should not have to be installed on every machine that is being visited. They use mobile code systems like Java and JVM (Java Virtual Machine) and classes get loaded at runtime via the network [16].

Mobile agents are operating instructions, or programs, that can be transmitted from a user's host to a remote host to perform specific tasks. Flexibility (or mobility) is the fundamental facet of mobile agents. A mobile agent appends its own performance, transfers to an alternative host, and remains to proceed at the break point [17].

Instead of transmitting data across the network, a mobile agent migrates to a geographically separated node, performs its task there and then returns to the original node (home node) bearing a result. Therefore, the mobile agent can utilize the bandwidth of the network more efficiently than one accessing the distributed database using a direct connection, especially when data transmission is the bottleneck of the task [18, 19, 20, 21]. Consequently, mobile agents reduce network traffic, overcome network latencies and enhance robustness and fault-tolerant capabilities of distributed applications [22, 23, 24].

It is important that mobile agents monitor the conditions of a network. The status of the network constantly changes in the Internet world. Therefore, a mobile agent, which is sensitive to the conditions of the network, can accomplish retrieval work more effectively [25, 26]. Along with the module which monitors network conditions, access to network status history can help mobile agents establish a static plan [27, 28]. It is very important to establish a static plan for mobile agents before mobilizing it. When mobile agents use past information about network conditions, system overhead, due to the reaction of mobile agents to the environment of the dynamic network, can be reduced. They can arrive at their destination more quickly in normal network traffic conditions, if they know the short cut. This means that the possibility of arriving at the destination in a timely manner increases. Planning the courses of mobile agents is called Mobile Agent Planning (MAP). MAP is one of the important techniques used to complete a given task efficiently.

### 2.4 Advantages of Mobile Agent Programming

The following are the primary advantages of mobile agents:

- They facilitate high quality, high performance, and economical mobile applications. Applications employing mobile agents transparently use the network to accomplish their tasks, while taking full advantage of resources local to the machines in the network. They process data at the data source, rather than fetching it remotely, allowing higher performance operation. They use the full spectrum of services available at each point in the network, such as GUI's for the user and database interface on servers. They make best use of the network as they travel [29].
- They enable the use of portable, low-cost, personal communications devices. Network support, including security, is contained in a lightweight server which manages the movement of agents in the network. Coupled with the sophisticated, self-contained programming model afforded by agents, this permits a small footprint to be achieved on user devices, without sacrificing functionality for the application.
- They permit secure Intranet-style communications on public networks. Security is an integral part of the Mobile Agent framework, and it provides for secure communications even over public networks. Agents carry user credentials with them as they travel, and these credentials are authenticated during execution at every point in the network. Agents and their data are fully encrypted as they traverse the network. All this occurs with no programmer intervention.
- They efficiently and economically use low bandwidth, high latency, error prone communications channels. The agent network employs a store and forward mechanism to transfer agents between nodes. This is well-suited to the problematic nature of many communications channels, especially in the mobile arena. Queuing and persistent

checkpoints enhance this further, to the point that agents can use such channels with no degradation in reliability or response. Because the agent's data processing takes place locally at the source, the network has no effect on the agent as it executes.

Marzo et al. [30] discussed many agent mobility approaches focusing on messengers. They said that mobile agents are mobile threads of execution and collaboration. The construction of novel couriers (messengers) at run time, and development of a cluster of couriers for courier alliance are incorporated in the flexibility facets of couriers. These characteristics appear relatively parallel to multi-agent systems (MAS) with spawning. Conversely, exploitation of a communal memory and depending on it for their operation of couriers is a major dissimilarity - insupportable in MAS as it suggests a resilient constraint on their independence.

## 2.5 Agent Cloning

Agent cloning is creating and activating a new agent with exactly the same capacities, capabilities as a possible response to an agent overload. Agent overloads are due, in general, either to the agent's limited capacity to process current tasks or to machine overloads. Other approaches to overloads include task transfer and agent migration. Task transfer, which occurs when overloaded agents locate other agents which are lightly loaded and transfer tasks to them, is very similar to processor load balancing. Agent migration, which requires that overloaded agents or agents that run on an overloaded machine (these loads are different but may correlate) migrate to less loaded machines, is closely related to process migration and to the recently emerging field of mobile agents [31]. A main difference between load balancing and agent cloning is that while the first explicitly discusses machine loads and agent migration, the latter, in addition, considers a different type of load - the agent load.

Cloning is a superset of task transfer and agent migration; it includes them and adds to them as well. Cloning does not necessarily require migration to other machines. Rather, a new agent is created on either the local or a remote machine. Note that there may be several agents running on the same machine, and having one of them overloaded does not necessarily imply that the others are overloaded (although we expect some correlation between overloads). Agent overload does not imply machine overload, and therefore local cloning (i.e., on the same machine) may be possible. As mentioned in the load balancing literature [32], within a distributed system there is a high probability of having some of the processors idle, while others are highly loaded. Cloning takes advantage of these idle processing capacities.

To perform cloning, an agent must reason about its own load (current and future) and its host's load, as well as capabilities and loads of other machines and agents. Accordingly, it may decide to create a clone, pass tasks to a clone, merge with other agents, or die. Merging of two agents or self-extinction of underutilized agents is an important mechanism to control agent proliferation with resulting overload of network resources. Detailed consideration of this problem, however, is outside the scope of this paper.

To avoid communication overhead in trying to access and reason about remote hosts, reasoning regarding cloning begins by considering local cloning. When this is found infeasible or non-beneficial, the agent proceeds to reason about remote cloning. If remote cloning is decided upon, an agent should be created and activated on a remote machine. Assuming that the agent has an access and a permit to work on this machine, there may be two main methods of performing this cloning:

- Creating the agent locally and letting it migrate to the remote machine (similar to a mobile agent).
- Creating and activating the agent on the remote machine.

While the first method requires very little on the part of the remote machine, it requires mobilization properties as well as additional local resource consumption. The second method, while avoiding mobilization and local resource consumption, requires that a copy of the agents' code be located on the remote machine. Similar requirements also hold for mobile agent applications [33, 34], since an agent server or agent dock is required. Nonetheless, the amount of this code is small.

Since the agent's own load and the loads of other agents vary over time in a non-deterministic way, the decision of whether and when to clone is non-trivial. Prior work has presented a model of cloning based on prediction of missed task deadlines and idle times on the agent's schedule in the RETSINA multi-agent infrastructure [35, 36].

Suppose a clone has been created and activated. Several questions remain with respect to this clone.

These regard its autonomy, tasks, lifetime, and access to resources. Autonomy refers to an independent versus a subordinate clone. Having been created and activated, an independent clone is not controlled by its creator. Therefore, such a clone continues to exist after completion of the tasks provided by its initiator agent. Hence, a mechanism for deciding what it should do afterward is necessary. Such a mechanism must allow the clone to reason about the agent and task environment, and accordingly decide whether it should continue to work on other tasks (if necessary and the computational resources allow), merge with others, or perform self-extinction.

A subordinate clone will remain under the control of its initiator. This prevents the complications arising as in the independent clone case (i.e., it is not necessary to decide what to do after the tasks delegated to the clone are accomplished). However, in order to manage a subordinate agent, the initiating agent must be provided with a control mechanism for remote agents. Regardless of the details of such a mechanism, it requires additional communication between the two agents, thus increasing the communication overhead of such a cloning method and the MAS's vulnerability to communication flaws. In addition, control of other agents is a partially centralized solution, which might violate the reason for using MAS in the first place.

### 2.6 Cloning Initiation

An agent should consider cloning if:

- It cannot perform all of its tasks on time by itself or decompose them so that they can be delegated to others.
- There is no lightly loaded agent that can receive and perform its excess tasks (or subtasks when tasks are decomposable).
- There are sufficient resources for creating and activating a clone agent (on either the same machine or a remote one).
- The efficiency of the clone agent and the original agent is expected to be greater than that of the original agent alone.

The necessary information used by an agent to decide whether and when to initiate cloning comprises parameters that describe both local and remote resources. In particular, the necessary parameters are as follows:

- The CPU and memory loads, both internal to the agent (which results from planning, scheduling and task execution activities of the agent) and external (on the agent host and possibly on remote hosts).
- The CPU execution speed (measured using standard methods e.g., MIPS), both local and remote. The load on the communication channels and their transfer rate, both local and remote.
- The current queue of tasks, the resources required for their execution, and their deadlines.
- The future expected flow of tasks.

To acquire the above information, an agent must be able to read the operating system variables. In addition, the agent must have self-awareness at two levels, at agent internal level and at MAS level. Internal self-awareness should allow the agent to realize what part of the operating system retrieved values are its own properties (i.e., agent internal parameters). System-wise self-awareness should allow the agent to find, possibly via middle agents [37], information regarding available resources on remote machines. Without middle agents (e.g., matchmakers), servers that are located on the remote hosts can supply such information on request.

### 2.7 Agent Spawning

Agent spawning is similar to agent cloning but it includes creating and activating a new agent with different capacities and capabilities [38, 39, 40]. While agent cloning is a possible response of an agent to overloads, agent spawning includes, in addition, consideration of the data transfer necessary for task execution and it relaxes the requirement of creating an identical copy of the original agent. Thus, spawning further enhances efficiency of network utilization and reduction of communication and computation loads.

A proxy agent is obligated to take its duty, contemplate the present and forthcoming transmission of the load of its host and other machineries, suggested by Shehory et al. [41]. The reasoning for spawning agents begins with distinguishing the kinds of issues that are needed to be fixed. There are two possible issues: (1) volume and competence, in other words, an agent is overloaded with a complicated and heavy task that requires time and skills, and (2) bulky material transmission necessities, in other words, network is overloaded by many requests for data from different sources.

In the event of encumbers, the main agent is triggered to (1) divide a complicated task into subtasks and (2) spawn light version of agents and delegate these subtasks to the spawned agents. This operation would effectively reduce the time required to complete the task by one agent. The reduction in time equals the required time by one agent divided by number of spawned agents, e.g. if a task take 10 minutes by one agent and in case this agent is able to spawn 100 light version agents to complete the task, then the time will be 10 /100, which equals 6 seconds.

There are two types of spawning, local and remote. However, remote spawning could be initiated when local spawning is unworkable due to overloaded network in sending the agents out as described in the second issue and when remote spawning is unquestionable. There are two approaches to execute remote spawning: (1) generating a proxy in the vicinity, and allowing it to transfer to a remote machine; or (2) generating and actuating the proxy on the remote machine.

When an agent is overloaded (i.e., it cannot complete the subtasks in its subtask waiting list before their respective deadlines or it has too many neighbors to keep with), the agent creates a new agent to handle parts of its load. The agent has two options, namely cloning or spawning an agent.

Specifically, for a single agent, spawning is triggered when the task load exceeds the agent's ability to complete on time, given the agent's current status and resource level. In this condition, the agent spawns some new agents and assigns the most beneficial tasks and corresponding resources to them. These spawned agents are subordinates of the original agent, but they cannot establish relations with other agents. When the spawned agents complete the assigned tasks, they become idle and if they are idle for a pre-defined period, i.e. when no more subtasks need to be completed, they are terminated by the main agent to save relation management load.

On the other hand, cloning happens when an agent has too many neighbours, which means that the agent has a heavy overhead for managing relations with other agents. In this situation, to avoid possible communication congestion, the agent clones an agent which has the same resources as itself, and assigns some neighbours to the cloned agent. The main agent keeps a peer relation with the cloned one. Contrary to the spawned agent, the cloned agent cannot be destroyed by the main agent. Instead, the main and cloned agents rejoin together, once the total number of neighbours is less than a pre-defined threshold.

### 2.8 Why is spawning necessary?

While agent cloning is a possible response to overloads, agent spawning includes, in addition, consideration of the data transfer necessary for task execution and it relaxes the requirement of creating an identical copy of the main agent. Thus, spawning increases the ability of a multi-agent system to perform tasks and reduces network congestion, enhances efficiency of network utilization and reduces communication and computation loads. We can say that spawning improves agents' performance and by using a spawning mechanism they can complete their tasks sooner.

## 3. DISCUSSION ON THE REVIEW

In this paper, we discuss about software agent technology and software agent cloning and spawning. While there are few research papers on agent spawning, many researchers Shehory et al. [41] have raised and pointed out some problems on agent spawning or cloning. These problems are:

1. What is are the optimal conditions for spawning or cloning to maximize the benefits of these two processes?
2. What is the optimal number of agents that should be spawned or cloned?
3. Do we have to spawn as many agents as we can or and if we do so, will it affect the network utilization?
4. Does cloning or spawning negatively affects the performance of a network?
5. What security issues are there that should be considered if we spawn or clone an agent and transfer it to another host in a network?

## 4. A PROPOSED AGENT SPAWNING AND MOBILITY FRAMEWORK

The preliminary theory of our proposed framework is based on a heavyweight agent, α, which is tasked to access public information from many different sites. Its design gives it the ability to reason on knowledge, actions, plans and communication so much so that transferring itself to the sites would severely affect its performance due to bandwidth constraints. Agent  then decides to spawn n agents, a1, a2, . . ., an, each of which is conferred specific functions and plans to access and gather the public information from the sites such that its overall plan is fulfilled. Figure 1 illustrates the spawning and mobility scenarios.

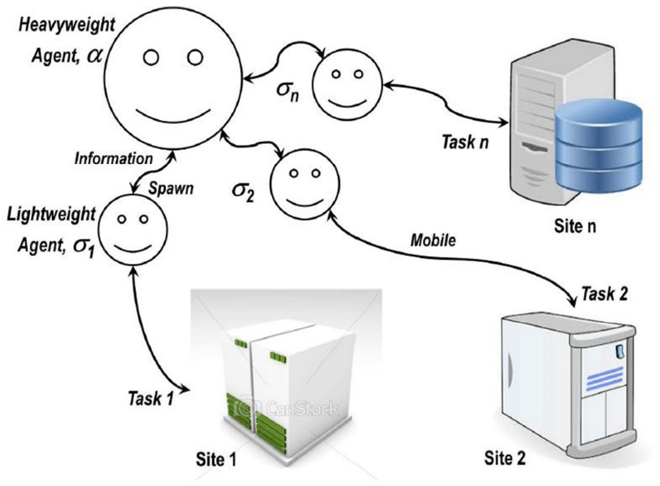

*Figure 1 The Spawning And Mobility Concept*

Being familiar with the nature of the issue that ought to be resolved, reasoning about spawning involves large information transfer requirements or capacity and capability overloads. Agent overloads are due to either the agent's limited capacity to process current tasks, or machine overloads. The following scenario explains the agent's actions in case of these two problems occur.

An agent, which is encumbered, should permit other agents that are capable of executing tasks. As soon as every agent is encumbered, they should permit newcomers to execute the tasks and make use of unexploited assets. On the other hand, relocating to other hosts is permissible for agents. As soon as new tasks are assigned to the agents and they are not competent, newcomers that are competent enough to execute the new tasks ought to be shaped and actuated. In view of that, there must be a specific competence server accessible to users and agents - localizing constituents for the required expertise of an agent, and generate agents by means of these constituents. If anticipated volume of remotely-situated data required for performing a task is bulky and is matched to the number of agents, the agents that wish to accomplish the task have to transfer to the location of the data, or agent at this location should be spawned. Fig. 2 summarizes this scenario.

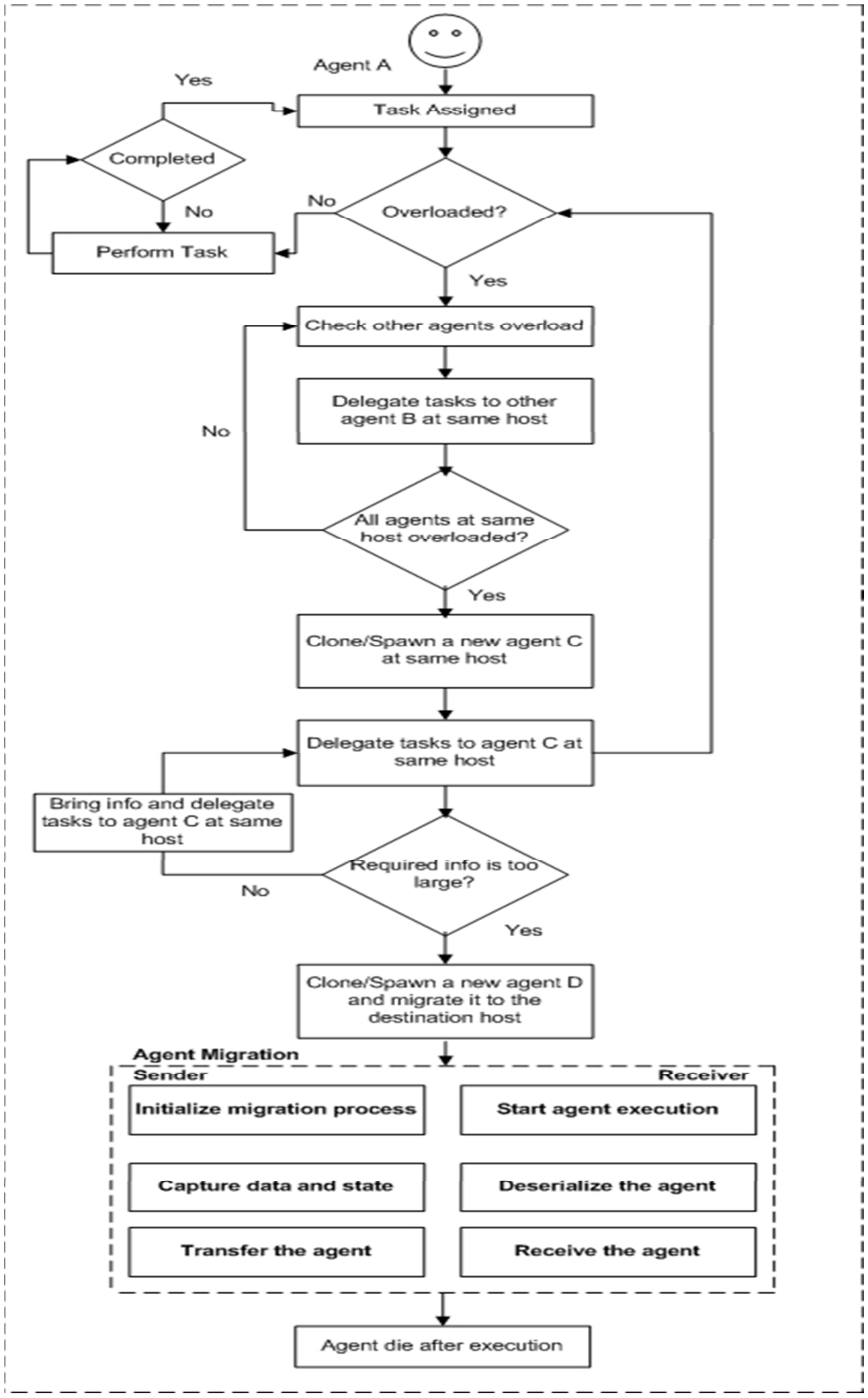
Agent A
Task Assigned
Yes
Completed
No
Perform Task
No
Overloaded?
Yes
Check other agents overload
Delegate tasks to other agent B at same host
No
All agents at same host overloaded?
Yes
Clone/Spawn a new agent C at same host
Delegate tasks to agent C at same host
Bring info and delegate tasks to agent C at same host
Required info is too large?
No
Yes
Clone/Spawn a new agent D and migrate it to the destination host
Agent Migration
Sender
Receiver
Initialize migration process
Start agent execution
Capture data and state
Deserialize the agent
Transfer the agent
Receive the agent
Agent die after execution

*Figure 2 The Spawning And Mobility*

The framework is described as follows:

If agent 'A', which has been assigned to a task, is overloaded, it passes the task to another agent 'B' at the same host, otherwise, it continues with executing its task.

If all agents at the same host are overloaded, a new agent, 'C', is cloned or spawned at the same host with capabilities that commensurate with the task's complexity.

If the host itself is overloaded or the information that is required for the task is too large, an agent, 'D', is cloned or spawned with capabilities that commensurate with the task's complexity and is mobilized to the destination host.

For agent migration from the sender host to the destination host, there are three steps each host will do. For Sender:

1. Initialize migration process.
2. Capture data and state.
3. Transfer the agent.

For Receiver:

1. Receive the agent.
2. De-serialize the agent.
3. Start agent execution.

At the end of the execution, the agent dies.

## 5. A FRAMEWORK FOR DYNAMIC SPAWNING OF AGENTS

We propose a framework for dynamic spawning of agents, which is based on the rate of incoming tasks and a heavyweight agent's CPU load. The heavyweight agent continuously monitors its CPU and make a decision whether or not to spawn. If it is unable to spawn a new agent to execute a specific task, it only spawns a new agent and sends it to another host to execute that task.

The issues in agent spawning involve the following challenges; the required number of spawned agents for a task group, the division of a task group into actions and the number of actions given to a spawned agent. Additionally, Shehory et al. [41] claimed that there are necessary parameters the heavyweight agent should use to decide whether and when to initiate spawning. The necessary parameters are as follows:

- The expected ratio of raw data necessary for its tasks.
- The CPU and memory loads, both internal to the agent and external (on the remote hosts).
- The CPU performance, both locally and remotely.
- The load on the communication channels and their transfer rate, both locally and remotely.
- The current queue of tasks, the resources required for their execution and their deadlines.
- The future expected flow of tasks.

We choose these parameters for the spawning function in order to know how many agents should be spawned at a specific time. Figure 3 depicts a typical scenario which shows a queue of task groups received by the agent at a task rate, □, from which the agent estimates the duration of each task group, t, and decides the number of actions, N, for the spawned agents. Concurrently, it also monitors the CPU load, □, as another parameter for the decision to spawn.

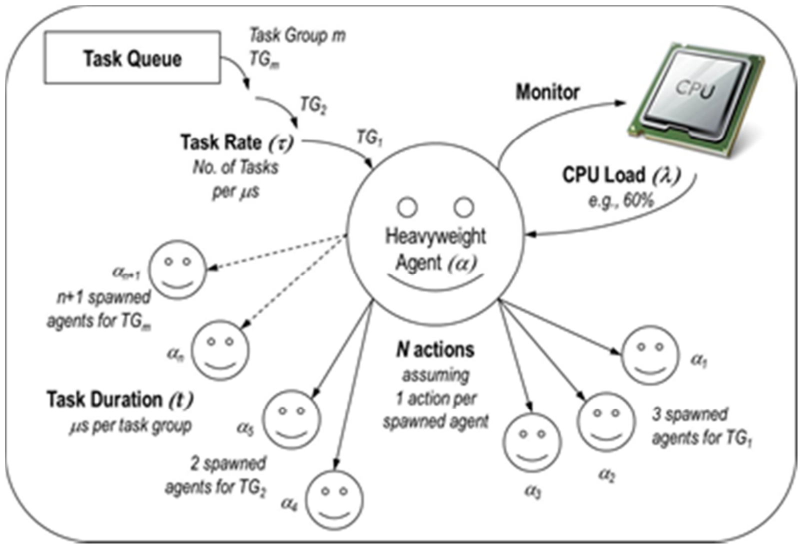

*Figure 3 Dynamic Spawning Of Agents*

Consequently, based on Figure 3, if σ is the spawning function, then the number of agents spawned is as follows:

If $\alpha_i$ is the number of the spawned agents, then:

$$\sigma : \alpha \times \{\tau, t, \lambda, N\} \rightarrow \alpha_i, (i \geq 1) \qquad (1)$$

i.e., the number of spawned agents is a function of the heavyweight agent, α, and the factors: τ, t, λ and N where,

τ = incoming task rate
t = duration of a task group
λ = CPU load
N = number of actions in a task group.

To acquire the above information, the agent must be able to read the operating system variables. In addition, the agent must have self-awareness on two levels, an agent internal level and a MAS level.

For this framework, we assume the following:

- Handle one task at a time: The heavyweight agent receives one task per unit time, handles it, and then receives the next one to start handling and so on.
- No port security issues.
- Spawn N agents for each task that has N actions.
- In a mobile agent system, the most common type of fault that can occur is that an agent suddenly disappears or is destroyed while moving from one node to another. This kind of fault is called agent crash (or simply, crash). So, we assume that the system is otherwise reliable (Fault-Tolerant Simulation of Message-Passing Algorithms by Mobile Agents)
- The transmitted mobile code runs on a new remote host.

## 6. CONCLUSION AND FURTHER WORK

In this review paper, we present an introductory concept of the research field in mobile agents. We discuss about agent architectures and classes, agent communication languages, agent cloning, spawning and mobility. We then propose a new model for agent spawning and mobility to solve the problem of agent overload. In our further work, we shall implement the framework utilizing the parameters for spawning and mobilizing optimal lightweight agent.

## 7. ACKNOWLEDGMENT

This project is sponsored by the Malaysian Ministry of Higher Education (MoHE) under the Fundamental Research Grant Scheme (FRGS) No. FRGS/2/2013/ICT07/UNITEN/02/1.

## REFERENCES

[1] Baumann J. and K. Rothermel. The shadow approach: An orphan detection protocol for mobile agents. In Int'l Workshopon Mobile Agents, 1998.

[2] White J. Telescript technology: Mobile agents. MIT press, 1997.

[3] de Krester O., A. Moffat, T. Shimmin, and J. Zobel. Methodologies for distributed information retrieval. In Proc. of the Eighteenth Int'l Conference on Distributed Computing Systems, pages 26–29, May 1998.

[4] Picco G., A. Carzaniga, and G. Vigna. Designing distributed applications with mobile code paradigms. In Proc. of the 19th Int'l Conference on Software Engineering, July 1997.

[5] Rus D., R. Gray, and D. Kotz. Autonomous and adaptive agents that gather information. In AAAI'96 International Workshop on Intelligent Adaptive Agents, Aug. 1996.

[6] Mahmoud, M. A., & Ahmad, M. S. (2015, August). A self-adaptive customer-oriented framework for intelligent strategic marketing: A multi-agent system approach to website development for learning institutions. In Agents, Multi-Agent Systems and Robotics (ISAMSR), 2015 International Symposium on (pp. 1-5). IEEE.

[7] Ahmed M., Ahmad M S, Yusoff M Z M, Modeling Agent-based Collaborative Process , The 2nd International Conference on Computational Collective Intelligence Technology and Applications (ICCCI 2010), pp. 296-305, ISBN:3-642-16692-X 978-3-642-16692-1, 10-12 November, 2010 Taiwan.

[8] Ahmed M., Ahmad M. S., and Yusoff M. Z. M., "A Collaborative Framework for Multiagent Systems." International Journal of Agent Technologies and Systems (IJATS), 3(4):1-18, 2011.

[9] Wooldridge, M. (1998). Agent-based computing. Interoperable Communication Networks, 1(1), 71-97.

[10] Mahmoud, M. A., Ahmad, M. S., & Yusoff, M. Z. M. "Development and implementation of a technique for norms-adaptable agents in open multi-agent communities." Journal of Systems Science and Complexity 29.6 (2016a): 1519-1537.

[11] Mahmoud, M. A., Ahmad, M. S., & Yusoff, M. Z. M. (2016b, March). A Norm Assimilation Approach for Multi-agent Systems in Heterogeneous Communities. In Asian Conference on Intelligent Information and Database Systems (pp. 354-363). Springer Berlin Heidelberg.

[12] Wooldridge, M. (1999). Intelligent Agents, The MIT Press.

[13] Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: theory and practice. The Knowledge Engineering Review 10(2), 115-152.

[14] Ahmed, M., Ahmad, M. S., & Yusoff, M. Z. M. (2009). A review and development of agent communication language. Electronic Journal of Computer Science and Information Technology: eJCIST, 1(1).

[15] Weiß, G. (2002). Agent orientation in software engineering. Knowledge Engineering Review, 16(4), 349–373.

[16] Mobile Agents an Introduction. (2014, December 1). Retrieved from http://www.cis.upenn.edu/~bcpierce/courses/629/papers/Concordia-WhitePaper.html#_Toc381690628

[17] Rongzhi Q. and S. Li, JMobile: A Lightweight Transparent Migration Mechanism for Mobile Agents, IEEE, 2008,pp. 1- 4.

[18] Athan A.and D. Duchamp. Agent-mediated message passing for constrained environments. In USENIX Mobile and Location-Independent Computing Symposium, 1993.

[19] Bandyopadhyay S. and K. Paul. Evaluating the performance of mobile agent-based message communication among mobile hosts in large ad hoc wireless network. In Proc. of 2nd ACM Int'l Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems, August 1999.

[20] 20 Miller L., J. Yang, V. Honavar, and J. Wong. Intelligent mobile agents for information retrieval and knowledge discovery from distributed data and knowledge sources. In Proc. of the IEEE Information Technology Conference, 1998.

[21] Sabnani K., T. L. Porta, T.Woo, and R. Ramjee. Experiences with network-based user agents for mobile applications. In Mobile Networks and Applications, 1998.

[22] Aridor Y.and M. Oshima. Infrastructure for mobile agents: Requirements and design. In Proc. on Int'l Workshop on Mobile Agents, 1998.

[23] Ghezzi C. and G. Vigna. Mobile code paradigm and technologies: A case study. In Int'l Workshop on Mobile Agents, April 1997.

[24] Wong D., N. Paciorek, and D. Moore. The promise of javabased mobile agents for unconstrained electronic commerce. Communication of the ACM, 42(3):92–102, 1999.

[25] Ohsuga A., Y. Nagai, Y. Irie, M. Hattori, and S. Honiden. Plangent: An approach to making mobile agents intelligent. IEEE Internet Computing, 1(4):50–57, 1997.

[26] Caripe W., G. Cybenko, K. Moizumi, and R. Gray. Network awareness and mobile agent systems. IEEE Communications Magazine, pages 44–49, July 1998.

[27] Moizumi K.. Mobile Agent Planning Problems. PhD thesis, Dartmouth College, 1998.

[28] Baek J., J. Yeo,G. Kim, andH.Yeom. Cost effectivemobile agent planning for distributed information retrieval. In Proc. on Int'l Conference on Distributed Systems, April 2001.

[29] Mobile Agent Computing. (2014, December 2). Retrieved from http://www.tryllian.com/mobile-agents/

[30] Di Marzo G., M. Muhugusa, and C. Tschudin. Survey of theories for mobile agents. Working paper, The Computing Science Center, University of Geneva, Switzerland, November 1995.

[31] Chess D., B. Grosof, and C. Harrison, "Itinerant agents for mobile computing," Tech. rep. RC 20010, IBM Res. Div., 1995.

[32] Shirazi B. A., A. R. Hurson, and K. M. Kavi, Eds. Scheduling and Load Balancing in Parallel and Distributed Systems, IEEE Comp. Soc. Press, 1995.

[33] Gray R. S. et al., "Mobile agents for mobile computing," Tech. rep. PCS-TR96-285, Dartmouth College, Comp. Sci., May 1996.

[34] White E., "Telescript technology: Mobile agents," General Magic White Paper, 1996.

[35] Sycara K. et al., "Distributed intelligent agents," IEEE Expert, Dec. 1996, pp. 3645.

[36] 36- Decker K., K. Sycara, and M. Williamson, "Intelligent adaptive information agents," Proc. AAA/ '96 Wksp. lntelligent Adaptive Agents, Portland, OR, 1996.

[37] Decker K., K. Sycara, and M. Williamson, "Middleagents for the Internet," Proc. CA/ '97, Nagoya, Japan, 1997.

[38] Alshaki, O. T., Ahmad, M. S., & Mahmoud, M. A. (2016, August). A new model of agent spawning and mobility. In Agent, Multi-Agent Systems and Robotics (ISAMSR), 2016 2nd International Symposium on (pp. 45-50). IEEE.

[39] Alshaki, O. T., Ahmad, M. S., & Mahmoud, M. A, and Mahmoud M. A. "Development of spawning and mobility model for heavyweight software agent." Agents, Multi-Agent Systems and Robotics (ISAMSR), 2015 International Symposium on. IEEE, 2015.

[40] Alshaki, O. T., Ahmad, M. S. "A Conceptual Framework for Agent Spawning ," in 2014 International Conference on Computational Science and Technology (ICCST), 2014, ISBN: 978-1-4799-3241-2

[41] Shehory O., K.Sycara, P.Chalasani, S.Jha,An Approach to gent Mobility and Resource Allocation, IEEE Commun. Mag. Jul. 1998, pp. 58-67.