Modified Diffusion Dynamic Load Balancing Employing Mobile Agents

MAGDY SAEB, CHERINE FATHY
Computer Engineering Department
Arab Academy for Science, Technology & Maritime Transport
Alexandria,
EGYPT
mail@magdysaeb.net, cherine_fathy@hotmail.com

Abstract: The escalating complexity and mobility of today's networks has led to the increased application of mobile agent paradigm. This paradigm helps to alleviate bandwidth limitations and supports disconnected operations that are both significant problems in wireless and mobile environments. On the other hand, load balancing is one of the important problems of computer heterogeneous networks. To address this problem, many centralized approaches have been proposed in the literature but centralization has proved to raise scalability tribulations.

In this work, we present a decentralized algorithm for diffusion dynamic load balancing based on mobile agent paradigm. We introduce the architecture of three types of agents employed to meet the requirements of the proposed diffusion load-balancing algorithm. We suggest a packet format for each type of agent as a data communication packet. Afterwards, we explain the different components of the simulator that we have developed to verify the effectiveness of the algorithm. Finally, the simulation results are discussed and a brief conclusion is provided.

Key-Words: Mobile agent, routing agent, load balancing, diffusion, data communication.

1. Introduction

The nature of today's data communication networks has rendered traditional load balancing solutions out of place. The traditional methods based on client-server paradigm rely on centralization or weak decentralization. In order to take a load balancing decision, information regarding the system processor state must be collected. This job is usually performed employing several approaches. First, a coordinator is utilized to send requests for information from all nodes periodically or, on the other hand, nodes disseminate their state information when their state changes. Moreover, algorithms with centralized component are less reliable because the failure of central component may cause the entire system to fail. In addition, a central component is potentially a bottleneck that limits load distribution activities.

An alternative approach is performed by using a number of nodes acting as coordinators. This is the decentralized approach; many nodes cooperate to take the load balancing decision. However, this method tends to give rise to high traffic rate wasting a large amount of network bandwidth. Another form of decentralization is based on the various diffusion algorithms. These algorithms are founded on the "locality principle" to achieve global load balancing. However, these algorithms make a lot of data exchanges in the network depending on the diffusion domain size.

These approaches are considered unsuitable for some of today's applications due to the following factors:

- Developments in wireless network and the wide spread of mobile devices which suffer from low bandwidth, unreliable connection, different transmission channels depending on their location, and diverse network addresses with each reconnection.
- The increasing size and heterogeneity of networks due to the high availability of cheap hardware.
- The proliferation of what is known as the "24-hours IP address".

To overcome some of these difficulties we resort to the mobile agent paradigm for diffusion-type load balancing.

Our proposed algorithm applies a number of different types of mobile agents in a cooperative way to fulfill the task of load balancing instead of single centralized component managing all load-balancing activities. Each type of agent implements one of the policies of load balancing algorithm. Moreover, the mobile agent paradigm supports the disruptive nature of wireless links and alleviates its associated bandwidth limitations. In addition, the use of mobile agents reduces network traffic since they carry the code of

their job and move to data to locally perform their intended computations.
The idea of using mobile agent in load balancing has been floating around for sometime in homogeneous telecommunication networks. Lipperts et al. [1] presented a mobile agent approach for load balancing in telecommunication. However, we present a new architecture of the mobile agents used to perform load balancing by taking the decision of migration of tasks in heterogeneous computer networks. The agent architecture is represented as a communication packet.
In the next section, we present an overview of mobile agent paradigm and the load-balancing problem. In section 3, we present the architecture of three types of agents used to fulfill the task of load balancing. These are information agents, location agents, and routing agents. We propose a packet format for each type of agents. In section 4, we explain the object model of the simulator that we have developed. In addition, we explain our proposed algorithm for load balancing. In section 5, the simulation results are discussed. The sensitivity of the algorithm to inter-arrival time and to the number of information agents is studied. Moreover, we compute the average response time of the cluster and the variance of the workload before, and after algorithm application. Section 6 provides a summary and our conclusions.

2. Mobile Agent and Load balancing Overview

A mobile agent is an executing program that can migrate, at times of its choosing, from machine to machine in a heterogeneous network. On each machine, the agent interacts with stationary service agents and other resources to accomplish its task [2].

Developments in wireless technology liberate network nodes from the constraint of being placed at a fixed physical location and enable the advent of the so-called mobile computing. The proliferation of mobile computing devices, which have the characteristics of low bandwidth and unreliable network connection, has lead to the increased use of mobile agent since it supports disconnected operations [3].

Mobile agent paradigm provides a better conservation of bandwidth since only the final result returns back to the client. When the server lacks one of the services, the mobile agent migrates to the server and performs the set of required operations locally. This, in turn, leads to reduction in total completion time. A mobile agent provides an effective means for overcoming network latency; it monitors the network latencies and continually moves to the network location that minimizes the average latency between itself and its clients.

Tveit [4] explained the difference between objects and agents; agents are regarded as a possible successor of objects since they can improve the abstractions of active entities.
Commonly, agents are classified based on their functionality. The classification is based on attributes specific to mobile agents: mobility, type of cooperation and level of interactions [5]. Rothermel et al. [6] provide another classification of software agents in terms of a space defined by the three dimensions of intelligence, agency and mobility.

During their nomadic life, mobile agents need to interact with other entities in order to carry out their jobs. In particular, they have to interact with the *local environment* of each arrival site. In addition, they interact with *other agents*, belonging both to the same application and to other applications. In this sense, Cabri et al. [7] have introduced the agent-local environment coordination and inter-agent coordination and have presented a taxonomy, which classifies the models of mobile agent coordination on the base of temporal and spatial coupling.

The current and future potential of mobile agents has led to a flurry of recent mobile code technologies. Mobile agent systems can be categorized in several ways. One of them is according to the programming languages that they support. Some systems allow agents to be written in multiple languages; many allow agents to be written in java only. This language is the most popular agent language. However, others allow agents to be written in some single language other than Java. Examples of these systems can be found in [2].

Fuggetta et al. [3] gives examples of mobile code domains of applications. To promote interoperability, some aspects of mobile agent technology have been standardized by OMG's MASIF [8]. In spite of the potential benefits of agent-oriented software engineering, this field is still immature. Wooldridge et al. [9] have pinpointed some of the drawbacks of building software using agent technology.

On the other hand, load-balancing problems arise in a distributed computing system when there is an unequal work allocation at the different processing elements in the system [2]. Load balancing improves performance by transferring tasks from heavily loaded computers where service is poor to lightly loaded computers, where the task can take advantage of computing

capacity that would otherwise go unused [10]. Performance typically improves, if the work can be equitably distributed among the elements [2].

Dynamic load-balancing algorithms differ in their degree of centralization. Algorithms can be centralized, hierarchical [11] and fully decentralized. As shown in [10], a more appropriate solution lies in fully decentralized algorithms.

3. Mobile Agent Architecture for Load Balancing

Network management is gaining mounting importance due to the wide spread use of computer networks. To overcome the problem of centralized management strategies and poor flexibility, code mobility is used in decentralized and flexible network management [12], [13], [14].

Mobile agents provide a support for dynamic load balancing with three key features [1]:

- They can move from one platform to another.
- They can move across heterogeneous platforms.
- They carry all application-specific code with them, rather than requiring pre-installation of that code on the destination machine. Most existing load-balancing systems are not as flexible as mobile agents are, since they lack one of the three key features.

3.1 Motivation

The motivation behind our architectural approach is as follow:

- The design of mobile agents in a way that can be implemented over the TCP/IP.
- Agents are taken as subclass of each other that simplifies the coding and provides low processing time.
- Ease of parsing of a packet representing an agent.
- Hierarchal structural design based on the principle of inheritance.
- The lifetime of agent is expressed by the number of hops that suits the heterogeneous environments.

The mobile agent can be seen as a communication packet transmitted over the network. Our model supports strong mobility, i.e., the code and execution state is transmitted upon agent migration. In this work, we propose a general format of the mobile agent packet as shown in Figure (1).

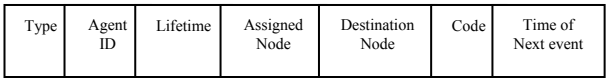

Fig.1: The Agent Packet General Format.

The basic agent packet consists of seven fields. The type field indicates the type of the agent: information agent or routing agent or location agent. Each agent has a unique identifier. The number of hops or lifetime field, i.e. the number of node visited before the agent terminates its life expresses the lifetime of the agent. The assigned node represents the current node that assigns the agent to the destination node. The code field represents the methods used by the agent to perform its task. The time of next event field indicates the time of arrival of agent at next destination node.

3.2 Basic Components and Assumptions

Our algorithm depends on three types of mobile agents. The domain of activity of these agents is restricted to one cluster. These types are information agent, location agent, and routing agent. In the following sections, we provide a description for each type.

3.2.1 Information Agent (IA)

The information agent plays the role of information policy component of any dynamic load-balancing algorithm as described in section 2. The information agent operates in two modes. First, it monitors the state of each node in the network.

The agent migrates to the shortest next destination node when the node is lightly or moderately loaded. On the other hand, if the state of the node is heavily loaded, it switches to the second mode. In the second mode, it launches location agent in all nodes with x hops far from the current node. If the routing information of nodes with x hops far from the current node is not available, the information agent launches location agent in all nodes with x-1 hop far of the current node. If it is not found, continue decreasing the number of hops until it arrives to one hop. This is the situation at the beginning of the simulation; each node has only information about the directly linked node. Each information agent contains a table of the location agent it has previously created and their numbers. Figure 2 depicts the packet format of the information agent.

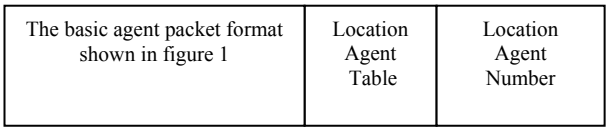

Fig. 2: The Information Agent Packet Format.

As shown in Figure (2), the information agent packet consists of the basic agent fields with additional two more fields. The location agent field is a table holding all the information of the location agent created by the information agent in the second mode. The Location agent number holds the number of location agent created by the information agent. Figure (3) shows the internal structure of the location table.

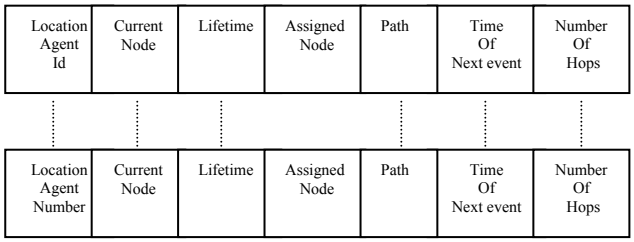

Fig. 3: The Location Agent Table Format.

Each entry in the location agent table consists of seven fields. The location agent id is a unique identifier for each location agent within the information agent. The current node represents the next destination of the location agent. The lifetime of the location agent is expressed in number of hops as described before. The assigned node represents the node that assigned the agent to the current node. The time of next event field indicates the time of arrival of agent at next destination node. The path field is a vector array holding the intermediate hops the location agent performs before arriving to its final destination. The number of hops field indicates the number of hops done so far by the location agent of the predefined number of hops. This field is used to access the path array to know the next intermediate node.

3.2.2 Location agent (LA)

The location agent plays the role of the location policy component of any dynamic load-balancing algorithm. The role of the location agent is to find suitable receiver partner for the overloaded-node that launched it. Figure (3) shows the internal structure of the location agent table.

3.2.3 Routing Agent (RA)

The routing agent is responsible of updating the routing table that resides at each node. It carries a routes vector table containing the communication cost from the assigned node to each other node in the cluster. This table has a lifetime measured by the number of the hops the routing agent is allowed to perform before updating the vector table. The routing agent plays an important role in informing each node in the cluster about the addresses of other nodes and if a failure of a node or a link is detected, it is the role of the routing agent to spread it over the cluster by copying the new table and migrating. However, it must be taken into consideration the propagation delay to move to each node in the cluster before the new table arrives to all other nodes. Figure (4) describes the general format of routing agent packet.

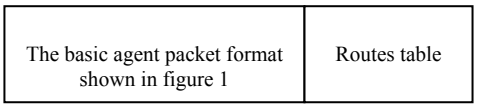

Fig. 4: The routing agent packet format

As shown in Figure (4), the routing agent packet consists of the basic agent fields with additional one field. The routes table field is a vector array holding the communication cost from the assigned node to all other nodes in the cluster.

3.2.4 The Fuzzifier

It is used to convert the load numerical value into fuzzy value. In our system, a max-membership fuzzifier is used (i.e., it selects the fuzzy set which has the maximum membership). In practice, a piecewise linear function, such as triangular or trapezoidal shape, simplifies the computation. Three fuzzy sets: Lightly Loaded, Moderately Loaded, Heavily Loaded are defined on the fuzzy variable load state. This component plays the role of transfer policy component. It is based on the output of the fuzzifier that the node is classified as sender or receiver or neutral.

4. The Simulation Study

In this section, we describe the object model of our simulator along with our load-balancing algorithm.

4.1 The Object Model of the Simulator

We have developed a simulator that considers all of the previously components and is based on the given assumptions. This was achieved using Borland C++ builder. It consists of sixteen basic classes. Figure (5) shows the object model of the simulator.

The simulator employs next-event time advance mechanism. The simulation clock is advanced to the time of the next event and the inactive periods are skipped [15].

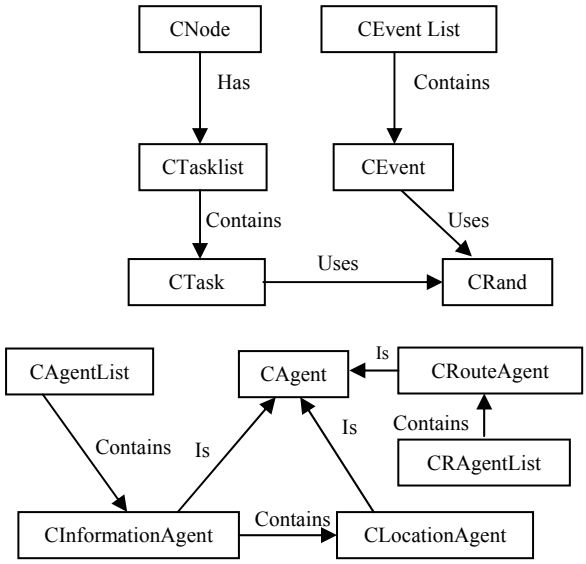

Fig. 5: The simulator object model.

The network model used is a cluster of, say 30 nodes, each of which has several attributes:

- A node identifier.
- A routing table initialized by only the communication information of current node to all other nodes in the cluster. The routing agent then completes this table. This table is the input to Floyd algorithm in order to compute all-pairs shortest path. A modification has been made to Floyd's in order to return a single two-dimensional array. The array x [i][j] contains the shortest communication cost value between i, j. Moreover, it contains a vector array of the shortest path between i, j, and the number of hops between i, j. This modification helps the agents during their movements since they need to know the required number of hops.
- A local queue, which is a task list of the tasks that will be executed locally.
- A flag indicating if there is a working agent or not. In our model no more than one agent is allowed to work in each node.
- A buffer containing the task for remote execution ordered according to their time of migration. In our model the last arrived task is selected for migration.
- Load index, which is the CPU queue length and a load description obtained from the fuzzifier, which defines the state of the node sender or receiver.

4.2 Dynamic Load balancing Algorithm

In this section, we propose a fully decentralized load-balancing algorithm based on mobile agents. Our algorithm is sender-initiated algorithm, i.e., an overloaded (sender) node initiates load-balancing activity trying to send a task to an under loaded node (receiver). The algorithm allows only the migration of non-preemptive tasks.

ALGORITHM MOBAGNTLDBALNC

[Given a cluster of nodes_num nodes, each is processing m tasks; three types of mobile agents are created to heuristically redistribute the load on these nodes in order to minimize the average response time and the load variance of this cluster of nodes.]

Input: nodes_num [number of nodes in the cluster], events_num *[number of events], pratio [ratio of cluster node number nodes_num], lifetimeinfo [the maximum number of nodes visited by information agent or location agent before its life is terminated], lifetimerout [the maximum number of nodes visited by routing agent before it update the movable route table, x [The number of hops to the next destination], β1 [Parameter of exponential distribution], β2,* α [Parameters of weibull distribution].

Algorithm Body:

Begin Algorithm
Generate a cluster with nodes_num nodes;
Generate the topology using the connectivity matrix of each node randomly;
Populate the routing table resident at each node with the communication cost of the directly linked nodes;
For i:= 1 to events_num
begin
Generate arrival event with y interarrival time where:

$$y \varepsilon \mathbf{Z}^{+} \mid \beta 1,\ y \sim \text{expo}\ (\beta 1);$$

Determine the arrival event destination node randomly;
Insert the arrival event into the event list ordered by the time of next event;
end;
Generate r routing agents where:

$$r \varepsilon \mathbf{Z}^{+} \mid \text{nodes\_num},\ r = \lceil \text{pratio} * \text{nodes\_num} \rceil\ ;$$

For i: =1 to r
begin
Set the routing agent parameter (Type, id, lifetime=lifetimerout, current node, assigned node);
current_routing_agent_list.Add_agent(routing agent);
end;
number= current_routing_agent_list.Return_agent_number();
For i: = 1 to number
begin
Each routing agent uploads the routing information of the current node;
Calculate the next destination node giving the number of hops allowed on each movement and the assigned node to prevent looping;
Insert event arrival of routing agent into the event list;
end;
While (The event list is not empty) Do
begin
Pop an event from the top of the list and checks its type;
If (Event Type=arrival event)
begin
Generate a task with Z service time where:

$$z \varepsilon \mathbf{Z}^{+} \mid \alpha\ \&\beta 2,\ z \sim \text{Weibull}\ (\alpha, \beta 2);$$

If (the processor is idle)
begin
Calculate the task's departure time;
Insert departure event into the event list;
end;
Else **begin**

```
                    Add the task to the node's local queue;
                    Load= Load + the task's length;
                end;
   end;
If (Event Type= departure event)
   begin
          If (The current node. load >0)
          begin
          Get a task from the queue and put it on the processor;
          Load=Load - the task's length;
          Calculate its departure time and insert departure event into the event list;
          end;
   end;
If (Event Type=migration event)
    begin
         Select the migrated task from the assigned machine buffer;
          If (The processor is busy)
             begin
                  Add the task to the queue of the destination;
             end;
          Else if (The processor is idle)
                 begin
                      Put the migrated task immediately on the processor;
                      Calculate the departure time and insert departure event in the event list;
                 end;
    end;
If (Event Type= arrival of information agent)
   begin
        If (current node. working agent =1)
           begin
                Terminate immediately the lifetime of the information agent according to the assumption done before;
            end;
        Else begin
                 Current node. working agent=1;
                 Check the load description of the current node;
                  If (Load status=lightly loaded or moderately loaded)
                 begin
                 If (Information agent's lifetimeinfo>0)
                    begin
                    The information agent moves to the next shortest destination with one hop far from the current node;
                    end;
                 end;
                 If (Load status=heavily loaded)
                    begin
                    for i:= 1 to nodes_num
                    begin
                    if (node i is x hop far of the current node)
                        begin
                         Launch location agent;
                         Copy intermediate x hops to destination into location agent's path array;
                         Add the location agent to information agent's location agent table.
                         If( x >1)
                  Insert event with event type location agent in transit to the first hop in the path array into the event list;
                         Else
                         Insert arrival event of location agent for each agent into the event list;
                         end;
                     end;
                    end;
end;
end;
If (Event Type= arrival of location agent)
   begin
         If (current node. working agent =1)
            begin
               Terminate immediately the lifetime of the information agent according to the assumption done before;
            end;
          Else
          begin
               Current node. working agent=1;
               Check the load description of the current node;
               If (Load status=lightly loaded)
               begin
               Return an acknowledgement to the assigned node with the current node number;
               end;
               If (Load status=heavily loaded or moderately loaded)
                   begin
                        If (Location agent's lifetime>0)
                          begin
                               The location agent calculates the next destination node with x hops far of the current node;
                               The location agent copies the intermediate hops into its path array;
                               If( x >1)
                               Insert event with type location agent in transit to the next hop in the path array into the event list;
                               Else
                               Insert arrival event of location agent into the event list;
                          end;
                   end;
          end;
   end;
If (Event Type== arrival of an acknowledgment)
          begin
          Check the load state of the current node;
           If (the load state of the current node is still heavily loaded)
              begin
              Select the last arrived task and insert it into the buffer for remote execution;
              Then, insert a migration event into the event list;
              end;
   end;
If (Event Type= arrival of routing agent)
   begin
          Check the lifetimerout of the movable routes table;

       If (the routes table's lifetimerout has expired)
          begin
               The routing agent uploads the routing information of the current node and calculates the next shortest destination node with one hop far of the current node;
          end;
          Else begin
```

```
                The routing agent downloads the movable route table
                 into the corresponding row in the routing table that
                 resides in the current node;
                 Calculate the next shortest destination node with one
                 hop far of the current node;
                 end;
   end;
If (Event Type= arrival of a location agent in transit)
   begin
   Calculate the next hop from the path array in the location agent;
   If (the next hop is the final hop)
      begin
            Insert arrival of location agent event into the event list;
      end;
   Else begin
             Insert arrival of location agent in transit event
            into the  event list;
       end;
   end;
Migration ()
       begin
      Generate n information agent to start monitoring the state of
       the workload of the cluster where:
          n ε Z+ | nodes_num, n = ┌ pratio * nodes_num┐ ;
       For each information agent, select randomly a destination
       node;
       For each information agent calculate the time of arrival  at
       the destination node and insert arrival event of information
       agent into the event list;  Add the information agent to
       current_working_agent_list;
       end;
end;
Generate report ()
begin
       Compute the average response time of the cluster for different
       workload as well as the variance of the load;
end;
      End Algorithm.
```

5. Simulation Results

This section provides the simulation results of the appliance of the proposed algorithm. In addition, we present a comparison between the performance metrics chosen in case of applying the algorithm and without any load balancing to the study the percentage of performance improvement the system gains with our algorithm.

5.1 Performance Metrics

To evaluate the performance of our proposed approach, we have used two performance metrics: the average response time of the cluster and the variance of the load over the network.

Our objective is to minimize the average response time and the variance as well.

5.2 The average response time of the cluster & the variance of the workload

As shown in Figure (6), a comparison between the average response time of the cluster in case of applying the load balancing algorithm and the average response time with no load balancing. The average response time is measured for different workloads: 100, 200, 400, and 600 tasks. It is clear from the figure that with the application of the algorithm, a better average response time was achieved for all levels of load condition.

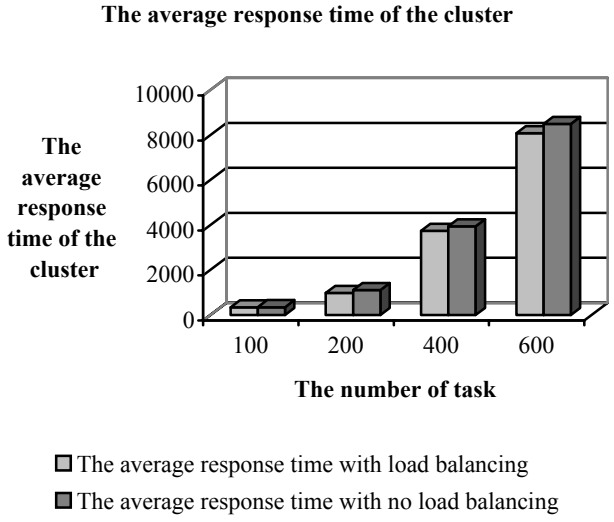

Fig. 6: The average response time of the cluster.

Figure (7) shows a comparison between the variance of the load over the cluster in case of load balancing and in case of no load balancing. The variance is measured for different workloads: 100, 200, 400, and 600 tasks.

As shown in figures (6) and (7), our proposed strategy has provided a noticeable improvement in the average response time of the cluster and also in the variance of the load.

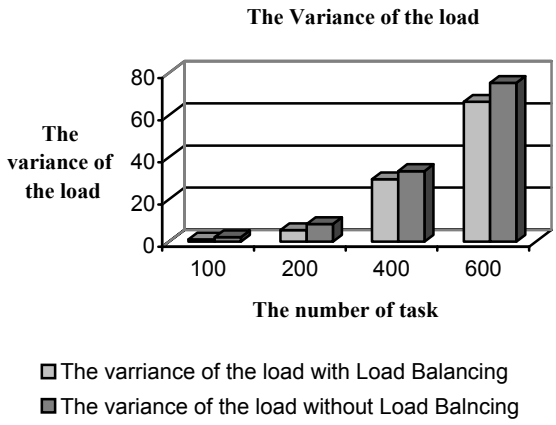

Fig. 7: The variance of the load over the cluster

5.3 The sensitivity of the algorithm to the number of information agent

Figure (8) shows the percentage of improvement in the average response time of the cluster for information agent number equals different percentage of the cluster size. We can conclude from figure (8) that increasing the number of information agents gives better performance when the number of tasks in the cluster is light or moderate. On the other hand, when the cluster is heavily loaded it is better to use a small number of agents. This is logical because of the computation overhead associated with an agent task and it is nonsense to make these computations when the cluster is already overloaded.

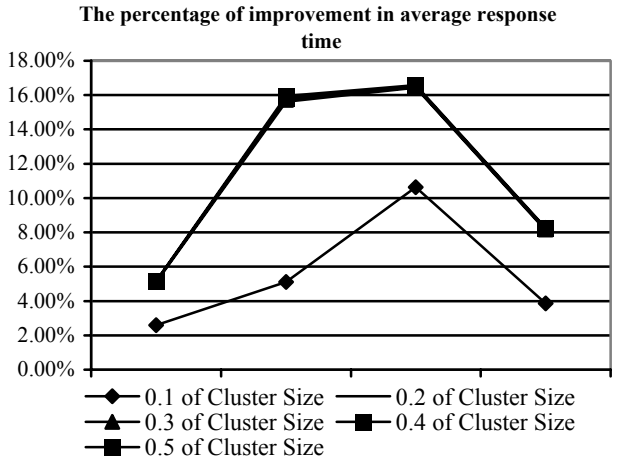

Fig. 8: The percentage of improvement in average response time depending on the percentage of the information agent of the cluster size.

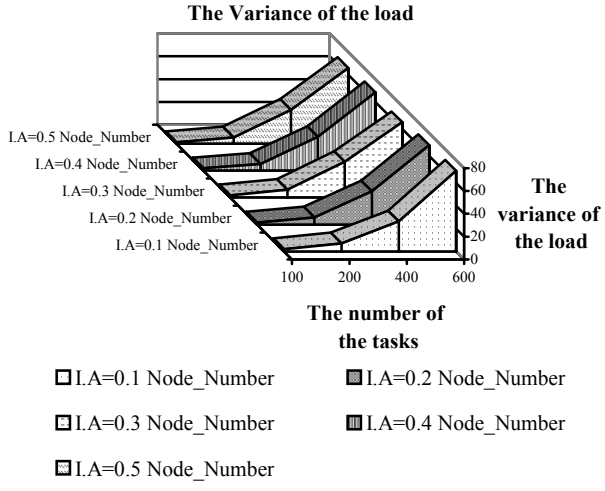

Fig. 9: The effect of number of agent and number of tasks on the variance of the load.

Figure (9) depicts the variance of the load over the cluster for different number of information agents taken as a ratio of the cluster size. Increasing the number of information agent improves the variance when the load on the cluster is moderate or light. However, on heavily loaded cases, the improvements are marginal.

5.4 The sensitivity of the algorithm to inter-arrival time

Figure (10) shows the sensitivity of our model to the increase of inter-arrival time for 100 tasks. When there are long gaps between the arrivals, the probability of a node moving to the receiver state increases. Thus, the probability of the sender–initiated policy locating a receiver increases with increasing variance in inter-arrival times. Thus, the average response time improves appreciably.

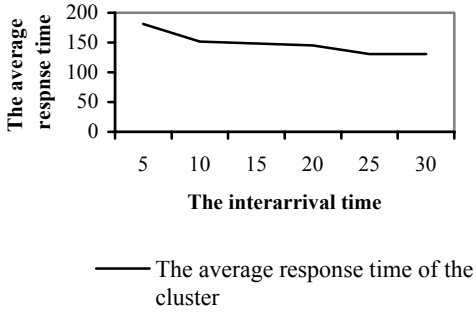

Fig. 10: The average response time of the cluster.

5.5 Discussion

Our strategy starts as diffusion strategy since at the beginning; the routing information available at each node is merely the information regarding the direct links. When the routing agent starts working and completes the routing information that resides at each node, the agent starts to make numerous hops and spread the workload far of the overloaded node. In this work we have adopted diffusion load balancing in the sense of locality principle but we achieved using mobile agent. This is the reason we call our algorithm a "modified" diffusion algorithm. The agent lifetime is measured in number of hops since measuring lifetime in time units is not a good indication in case of high communication cost where the timeout period may expire before agent can do its computation and achieves the decision to complete its job. Allowing only one agent per node has added to the stability of the algorithm.

6. Summary and conclusion

The development in wireless communication and mobile devices has led to the increased use of mobile agents with their great advantage in supporting low bandwidth and unreliable network connection. Load balancing has becoming a vital issue due to the increased size of computer networks and the amount of information that need to be managed. In this work, we have presented the following

- A decentralized approach for dynamic load balancing based on mobile agents is presented.
- We have proposed a packet format for the information agent, the location agent, and the routing agent.
- A simulator has been developed to simulate the problem. We have shown, through simulation, the validity of the algorithm to improve the variance of the load in the cluster and in providing better average response time.
- The sensitivity of the model to increasing the number of information agent and accordingly increasing the number of location agents is investigated. The increase of the number of agents provides a better average response time and smaller variance of the cluster's load when the cluster's load is light or moderate. In heavily loaded situation, increasing the number of agent negatively effect the situation.
- Finally, we have discussed the sensitivity of the algorithm to the increase of inter-arrival time and we have shown that increasing the arrivals gaps provides better average response time.

In this work, we have adopted diffusion load balancing, however we have modified it to allow variable diffusion domain. This modified diffusion load-balancing algorithm offers an alternative to client-server-based solutions with appreciable bandwidth savings. Moreover, since this technique requires relatively short time to implement, we believe that the method is clearly superior to client-server paradigm. The method lends itself to wireless data communication applications. Future work includes prototyping on a small-scale network to validate the simulation results.

References:

[1] S. Lipperts and B. Kreller, "Mobile agents in telecommunications networks: a simulative approach to load balancing," Proc. 5th Intl. Conf. Information Systems, Analysis and Synthesis, ISAS'99, 1999.

[2] Robert Gray, David Kotz, Saurab Nog, Daniela Rus and George Cybenko, "Mobile Agents: Motivations and state -of -the art systems," Technical report PCS-TR00-365, April 2000.

[3] Alfonso Fuggetta, Gian Pietro Picco and Giovanni Vigna, "Understanding Code Mobility," IEEE Transactions on Software Engineering, Vol 24, No.5, May 1998.

[4] Amund Tveit, "A survey of Agent Software Engineering," First NTNU CSGSC, May 2001.

[5] Niraj Joshi and VC Ramesh, "On Mobile Agent Architectures," Technical Report, 1998.

[6] Kurt Rothermel and Markus Schwehm, "Mobile Agents," Encyclopedia of Computer Science and Technology,(A. Kent and J. G. Williams EDS.), M. Dekker, Inc., 1998.

[7] G. Cabri, L. Leonardi and F. Zambonelli, " Mobile Agent Technology: Current Trends And Perspectives," Congresso Annuale AICA 98, Napoli, November, 1998.

[8] Dany B.lange," Mobile Objects and Mobile Agents: the Future of Distributed Computing," Proceedings of the European Conference on Object-Oriented Programming, 1998.

[9] Michael Wooldridge and Nicholas R. Jennings " Pitfalls of agent- oriented development," Proceeding of the second international conference on Autonomous agents, pages 385–391, 1998.

[10]Niranjan G. Shivaratri, Phillip Krueger, and Mukesh Singhal, "Load Distributing for Locally Distributed Systems," IEEE Computer, Vol. 25, No. 12, pp. 33-45, December 1992.

[11] Sivarama P. Dandamudi and K. C. Michael Lo, "A hierarchical Load Sharing Policy for Distributed Systems," Proceedings of the 5th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 1997.

[12] Buchanan WJ, Naylor M, Scott AV, "Enhancing Network Management using Mobile Agents," Proceedings Seventh IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2000).

[13] A. Biesczad, B. Pagurek, and T. White, "Mobile Agents for Network Management," IEEE Communications Surveys, 1998.

[14] M. Baldi, S. Gai, and G. Pietro Picco, "Exploiting Code Mobility in Decentralized and Flexible Network Management," Proceedings of the First International Workshop on Mobile Agents, Berlin, April 1997.

[15] Averill M. Law and W. David Kelton, Simulation Modeling & Analysis, Prentice Hall International Inc., 1991.