Performance Evaluation of Mobile Agent-based Dynamic Load Balancing Algorithm

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 proliferation of heterogeneity of today's network and the rapid development in wireless networks has raised an urgent requirement for a new communication paradigm. The Mobile agent paradigm is a promising technique that addresses the client-server limitations in network management applications. This paradigm helps to alleviate bandwidth limitations and supports disconnected operations that are significant problems in wireless and mobile environments. On the other hand, load balancing is one of the important problems of heterogeneous computer 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 dynamic load balancing based on the mobile agent paradigm. We introduce the architecture of three types of agents that are employed to meet the requirements of the proposed load-balancing algorithm. We explain the functionality of each suggested type of agent. Moreover, we propose a data communication packet format for each type of agent. Subsequently, and based on the developed simulator, our results are discussed. In addition, we study the impact of the cluster size, the agent lifetime on the algorithm convergence times and the variance of the workload over the cluster.

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

1. Introduction

For large networks, conventional network management is based on Simple Network Management Protocol (SNMP). It gives network administrators the flexibility of managing the entire network from a single node. However, SNMP is often used in centralized network management environments. The main disadvantages of this approach are:

- Information flow bottleneck at the single management node.
- Lack of scalability.
- Excessive processing loads at the management node.
- Inefficient usage of network bandwidth by network management actions.

An alternative approach is distributed network management. In this technique, the centralized management strategy is replaced by interoperable management systems. Distributed management helps to solve some of the problems associated with centralized management. However, it still has some drawbacks such as limited scalability and complex coordination mechanisms between management stations [1].

As load balancing is one of the network management activities, traditional load balancing techniques suffer also from the conventional network management drawbacks of centralization and decentralization discussed before. One form of decentralized load balancing is based on the diffusion algorithms. These algorithms are founded on the "locality principle" to achieve global load balancing. In order to take load-balancing decision, information regarding the system processor load must be collected. The limited amount of load information available to the diffusion policies can sometime lead them to inefficient load balancing actions or even to stop executing because of the recognition of a local load balance, without the capability of recognizing global imbalance [2].

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 dynamic load balancing.

Mobile agents provide a support for dynamic load balancing since they can move across heterogeneous platforms and they carry all application-specific code with them, rather than requiring pre-installation of that code on destination machine [3]. 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 abstract goal of load balancing can be stated as follows:

"Given a collection of tasks comprising a computation and a set of computers on which these tasks may be executed, find the mapping of tasks to computers that results in each computer having an approximately equal amount of work"[4].

Typically, a dynamic load-balancing algorithm has four components: a transfer policy, a selection policy, a location policy, and an information policy [5].

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. The idea of using mobile agent in load balancing has been floating around for sometime in homogeneous networks [6,7]. Lipperts et al. [8] presented a mobile agent approach for load balancing in homogeneous telecommunication networks.

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 data communication packet. This approach can be easily implemented over TCP/IP networks.

In the next section, 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 3, we explain our proposed algorithm for load balancing. In section 4, the simulation results are discussed. In section 5, the impact of the network size and the lifetime of the agent on the convergence rate of the algorithm and on the normalized variance of the workload over the cluster are studied. Section 6 provides a summary and our conclusions.

2. Mobile Agent Architecture for Load Balancing

The mobile agent can be seen as a data communication packet transmitted over the network. In this work, we propose a general format of the mobile agent packet as shown in Figure 1.

2.1 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.

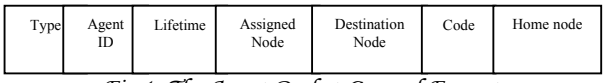

Fig.1: The Agent Packet General Format.

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 destination node is the next node to be visited by the agent. The code field represents the methods used by the agent to perform its task. The home node field is the node that created the agent. This field is used in order to insert all agent information in its home agent list. Each node contains a list of the agent created so far to maintain their information.

2.2 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.

2.2.1 Information Agent (IA)

The information agent plays the role of information policy component of any dynamic load-balancing algorithm. 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.

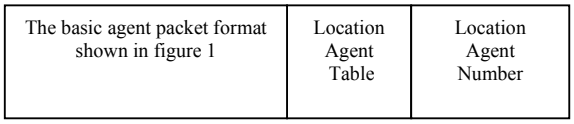

Fig. 2: The Information Agent Packet Format.

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.

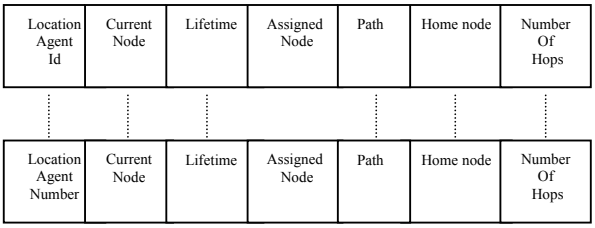

Fig. 3: The Location Agent Table Format.

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.

2.2.2 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.

2.2.3 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.

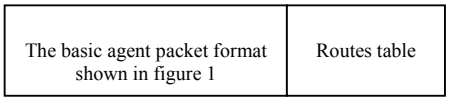

Fig. 4: The routing agent packet format.

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.

2.2.4 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).

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.

3 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. The algorithm details are shown in appendix I.

4. Simulation Results

This section provides a discussion of the simulation results of the proposed algorithm. We present an in depth comparison between a selected set of performance metrics with and without applying the suggested methodology.

4.1 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. We have chosen to use the normalized variance rather than its absolute value. Our objective is to minimize the average response time and the variance as well.

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

As shown in Figure 5, 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 ranging from 100 up to 1000 tasks. After 10 simulation runs, the average case is selected for each workload. It is clear from this figure that with the application of the algorithm, a better average response time was achieved for all load levels.

On the other hand, Figure 6 illustrates 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. After 10 simulation runs, the best case is selected for each workload out of the 10 simulation runs.

The improvement in average response time, as shown in from these two figures, has reached 22.65% in the best case and 17.4% in the average case.

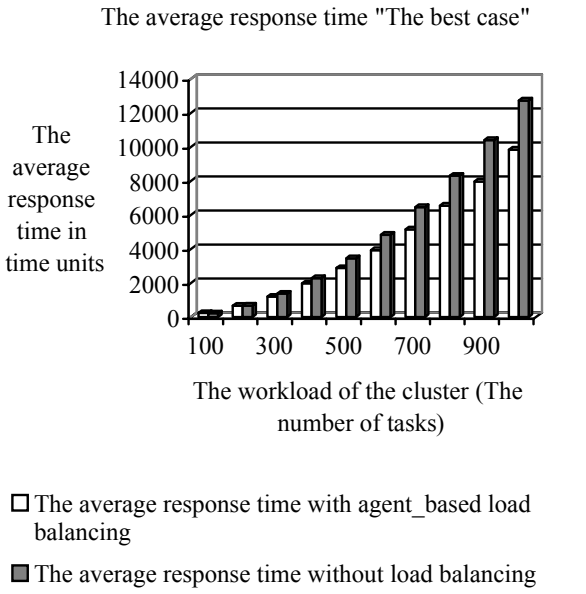

Fig. 5: The average response time of the cluster. "The best case is selected out of 10 simulation runs"

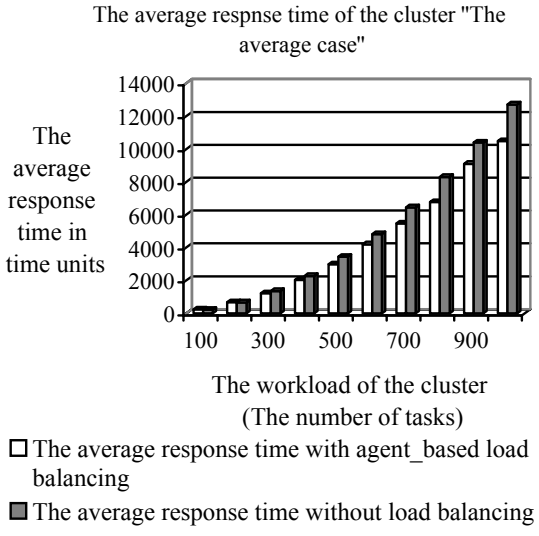

Fig. 6. The average response time of the cluster, "The average case is selected out of 10 simulation runs"

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 ranging from 100 up to 1000

tasks. After 10 simulation runs, the average case is selected for each workload.

Figure 8 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 average case is selected out of 10 simulation runs. The results show an improvement of 52.14% in the best case and 37.81% in the average case.

Consequently, We can conclude that our proposed strategy has provided a advantageous improvement in the average response time of the cluster and a noticeable improvement in the variance of the load over the cluster.

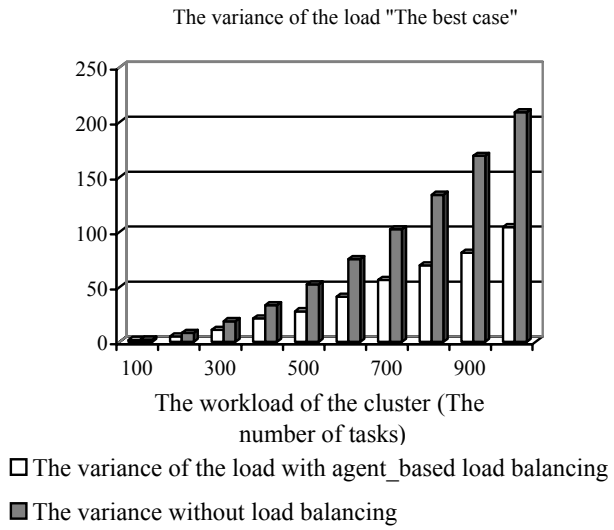

Fig. 7. The Variance of the load over the cluster, "The best case is selected out of 10 simulations runs"

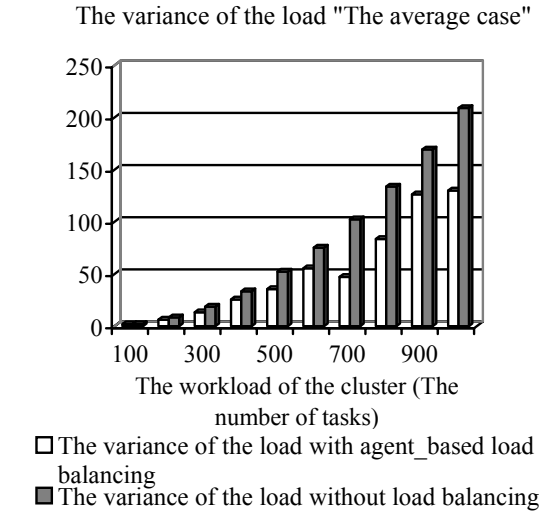

Fig. 8. The Variance of the load over the cluster," The best case is selected out of 10 simulation runs"

5. The impact of network size and the lifetime of the agent on the convergence rate of the proposed algorithm.

For a cluster of 30 nodes, workload consists of 1000 tasks and no load balancing variance V_{nlb} of 6.27586 without applying our mobile agent-based load-balancing algorithm. Figure 9 shows the time taken to converge to a normalized variance less than V_{nlb}. Obviously, The time increases with the decrease of variance.

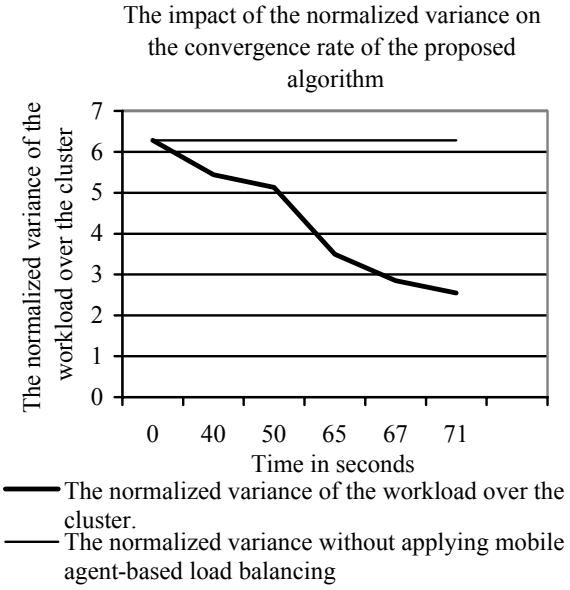

Fig. 9. The impact of the normalized variance on the convergence rate of the proposed algorithm.

Figure 10 depicts the relation between the cluster size and the time taken by the algorithm to converge to variance less then V_{nlb}. Increasing the cluster size will increase the number of information agents created to monitor the system and consequently increase the number of location agents created. Therefore, the algorithm takes more time to converge with the increase of the cluster size.

Figure 11 demonstrates the relation between the lifetime of the agent and the time taken by the algorithm to converge to variance less the V_{nlb}. Increasing the number of nodes visited by the agent i.e., number of hops will increase the communication and the computation overhead performed at each node. Hence, the algorithm will take more time to converge with the increase of agent lifetime.

Figure 12 illustrates the impact of agent lifetime on the variance of the workload over the cluster. Increasing

the number of hops the agent performs enables it to reach under-loaded node and this, in turn, provides better performance. Increasing the lifetime of the agent is an advantage to a certain point after which it becomes an overhead.

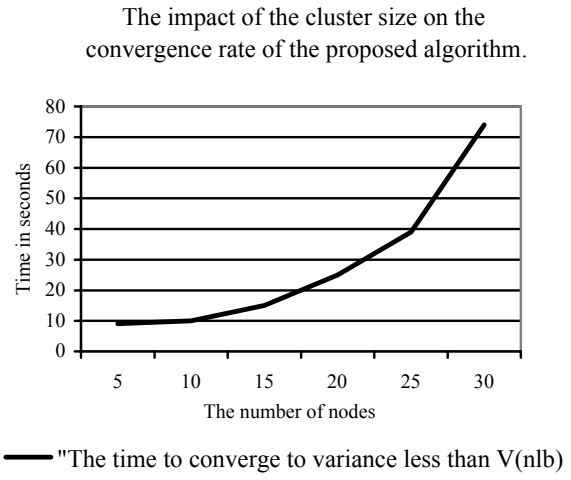

Fig. 10. The impact of the cluster size on the convergence rate of the proposed algorithm.

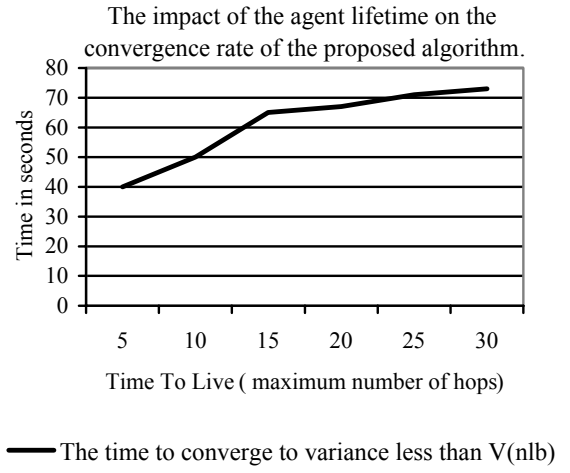

Fig. 11. The impact of the agent lifetime on the convergence rate of the proposed algorithm.

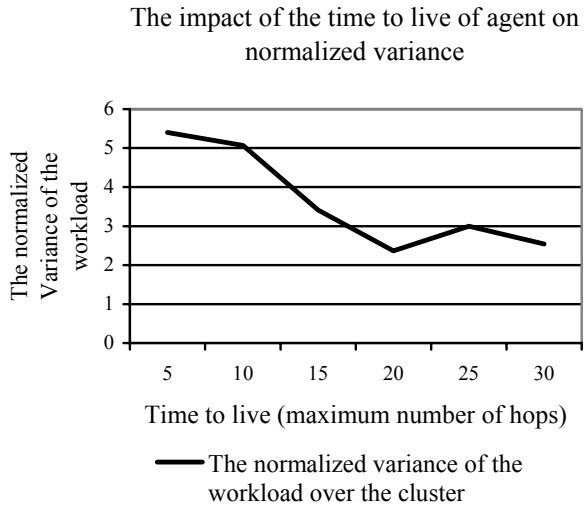

Fig. 12. The impact of the agent lifetime on the normalized variance of the workload.

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 become 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.
- A study of the impact of the cluster size and agent's "Time to Live" (TTL) on the convergence rate of the algorithm and on the normalized variance.
- Increasing the cluster size will increase the number of agents and therefore, the algorithm takes more time to converge with the increase of the cluster size.
- Increasing the number of hops the agent performs will increase the communication and the computation overhead done at each node and thus, the algorithm will take more time to converge with the increase of agent lifetime.

- Increasing the number of hops the agent performs enables it to reach under-loaded node and this gives better performance. To a certain point increasing the lifetime of the agent is an advantage after which it becomes an overhead.

This modified dynamic 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] Manoj Kumar Kona and Cheng-Zhong Xu, " A Framework for Network Management using Mobile Agents," International Parallel and Distributed Processing Symposium, Florida, 2002.

[2] Franco Zambonelli, "How to Improve Local Load Balancing Policies by distorting Load Information," 5th International Conference on High-Performance Computing (HiPC 98), Madras (India), IEEE CS Press, December 1998.

[3] 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.

[4] Jerrell Watts and Stephen Taylor, "A Practical Approach to Dynamic Load Balancing," IEEE Transactions on Parallel and Distributed Systems, Vol. 9, No. 3, March 1998.

[5] 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.

[6] Lu Lina, Liu Longguo, and Yang Xinyu, "An Agent-based Load Balancing Mechanism: PLRM Using Java," Proceedings of the 36th International Conference on Technology of Object-Oriented Languages and Systems, 2000.

[7] Sasa Desic and Darko Huljenic, " Agents-Based Load Balancing with Component distribution," Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2002.

[8] 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.

Appendix I: Formal Description of Algorithm

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], *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.*

Algorithm Body:

Begin Algorithm

1. Generate r routing agents where:
   $r \varepsilon \mathbf{Z}^{+} \mid$ nodes_num, r = ⌈pratio * nodes_num⌉ ;
2. For each routing agent
   a. Select home node h where the routing agent is created randomly where:
      $h \varepsilon \mathbf{Z}^{+} \mid$ nodes_num, h = random (nodes_num);
   b. Fill the routing agent packet fields with given and calculated data:
      Type ← 3; Home node ← h; current node ← h; assigned node← h; lifetime ← lifetimerout; Id ← RAid++;
   c. Each routing agent uploads the routing information of the current node;
   d. Calculate for the routing agent the next node to be visited with minimum communication overhead given the number of hops the RA is allowed to make to reach destination and given the current node and assigned node to prevent looping
      Next destination= find_ next hop (current node, Max_number_hops, assigned node);
   e. Insert event with type "arrival of routing agent" into the event list;
3. Generate n information agent to start monitoring the state of the workload of the cluster where:
   $n \varepsilon \mathbf{Z}^{+} \mid$ nodes_num, n = ⌈pratio * nodes_num⌉ ;
4. For each information agent
   a. Select home node h where the information agent is created randomly where:
      $h \varepsilon \mathbf{Z}^{+} \mid$ nodes_num, h = random (nodes_num);
   b. Fill the information agent packet fields with given and calculated data:
      Type ← 1; Home node ← h; current node ← h; assigned node← h; lifetime ← lifetimeinfo; Id ← IAid++;
   c. Calculate for the information agent the next node to be visited with minimum communication overhead given the number of hops the IA is allowed to make to reach destination and given the current node and assigned node to prevent looping
      Next destination= find_ next hop (current node, Max_number_hops, assigned node);
   d. current node← next destination;
   e. Insert a copy of IA packet into the information agent list residing in its home node & holding all information of the information agent created so far by this node;

f. Calculate time of arrival of IA at next destination;
g. Insert event with type " arrival of information agent" in the event list;

5. At arrival at each node, The IA checks the load description obtained by the fuzzifier of the CPU queue length;
   a. If the node status is "Heavilyloaded"
      i. Terminate the IA life as monitoring and switch to second mode;
      ii. The information agent launches Location Agent in all nodes with Max_hop far from the current node for which the routing information is available;
      iii. For each location agent, fill the packet fields with given and calculated data:
      Type ← 2; Home node ← IA.home node; assigned node← current node; lifetime ← lifetimeinfo;
      Id ← LAid++;
      iv. **If** (Max_hop =1) **begin**
      Calculate the arrival time of LA at next destination;
      Insert event with type "arrival of location agent" in the event list;
      Add LA packet to the LA Table in the IA packet;
      Insert the IA packet in the information agent list residing in its home node;
      end;

      else if (Max_hop >1) **begin**
      Copy the intermediate hops of LA to destination in its path array;
      Calculate the arrival time of LA at first node in the path array;
      Insert event with type "arrival of location agent in transit" in the event list;
      Add LA packet to the LA Table in the IA packet;
      Insert the IA packet in the information agent list residing in its home node;
      end;
   b. If the node status is "Lightlyloaded or Moderatelyloaded"
      If (IA. lifetime >0) **begin**
      Next destination= find_ next hop (current node, Max_number_hops, assigned node);
      Update data held in IA packet;
      Calculate time of arrival of IA at next destination;
      Insert the IA packet in the information agent list residing in its home node;
      Insert event with type " arrival of information agent" in the event list;
      end;

6. At arrival at each node, the LA checks the load description obtained by the fuzzifier of the CPU queue length;
   a. If the node status is "Heavilyloaded"
      begin
      Next destination= find_ next hop (current node, Max_number_hops, assigned node);
      Repeat step 5.a.iv again.
      end;

   b. If the node status is "Lightlyloaded or Moderatelyloaded"
      begin
      Send an acknowledgement to overloaded node that launched LA with current node as suitable partner to share load with;
      Terminate L.A life and remove it from L.A table;
      Insert event with type "arrival of acknowledgement" in the event list;
      Update data held in IA packet;
      Insert the IA packet in the information agent list residing in its home node;

7. At arrival of L.A in transit,
   a. If (number of hops done so far) >= Max_number_hops)
      begin
      Calculate the arrival time of LA at last node in the path array;
      Insert event with type "arrival of location agent" in the event list;
      Add LA packet to the LA Table in the IA packet;
      Insert the IA packet in the information agent list residing in its home node;
      end;
      else begin
      Calculate the arrival time of LA to next intermediate node in the path array;
      Insert event with type "arrival of location agent in transit" in the event list;
      Add LA packet to the LA Table in the IA packet;
      Insert the IA packet in the information agent list residing in its home node;
      end;

8. At arrival of acknowledgement,
   a. Check if the node is still overloaded;
   b. If yes, select the last task in the queue for migration;
   c. Calculate the time of arrival at remote node;
   d. Insert event with type "migration" in the event list;

9. At arrival of routing agent,
   a. Check the lifetimerout of the movable routes table;
   b. 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 Algorithm.