LEADER ELECTION ALGORITHM IN HYPERCUBE NETWOK WHEN THE ID NUMBER IS NOT DISTINGUISHED

Abstract

One of the most critical problems in distributed systems is Leader Failure. Distributed network becomes unstable without leadership. To solve this problem Leader Election algorithm is executed to give the leadership to other processor. The election process starts when one or more processor discovers that leader has failed, and it terminates when the remaining processors know who the new leader is. Election process use ID number to choose the new leader.

In this study we propose distributed leader election algorithm to solve leader failure in hypercube network when ID number is not distinguished. Performance is evaluated and analyzed. In a network of N nodes connected by a Hypercube network the proposed algorithm uses O(N) messages to elect a new leader in O(log N) time steps.

Key Words: Leader Election, Hypercube Network, ID Number, Time Complexity.

1. Introduction

One of the most fundamental problems in distributed systems is the leader failure.

This problem can be solved by Leader election Algorithms (LEAs). These algorithms moves the system from an initial state, where all the nodes are in the same computation state, into a new state where only one node is distinguished computationally (called leader).

Distributed systems are used to increase the computational speed of problem solving. These systems use a number of computers which cooperate with each other to execute some task. Control of distributed algorithms requires one process to act as a controller (leader). Leader is responsible to maintain the stability all time overall the network. If the Leader fail for any reason, new leader should be elected directly to recover from instability.

Leader election process is a program distributed over all nodes, it starts when one or more processors discover leader has failure, it terminates when remaining processors know who the new leader is.

LEAs are widely used in centralized systems to solve single point failure problem. For example, in client-server, LEAs are used when the server fails and the system needs to transfer the leadership to another station. The LEAs are also used in token ring. When the node that has the token fails, the system should select a new node to have the token.

In distributed systems, there are many network topologies like hypercube, meshes, ring, bus,…etc. These topologies may be either hardware processors, or the software processes embedded over other hardware topology. This study focuses on hypercube topology. This paper proposes a new election algorithm to solve leader failure automatically.

Election algorithms start when the leader failure is detected by one process or all processes at worst case. It terminates when all processes elect the new leader.

The organization of this paper will be as follows: Next Section presents Previous work. Section 3 describes the hypercube model structure and properties. Section 4 describes the proposed leader election algorithm. Mathematical proof for the time steps complexity is presented in section 5. Results conclusion and suggest future work in Section 6.

2 Previous Work.

Leader election algorithms have been studied by a number of researchers ([1], [2], [3], [5], [6], [9],[10], [11], [12], [13], [16], [17], [19], [21], [22], [24], [25], [26], [27], and [33]). In these studies, the researchers presented different methods to deal with the leader election algorithms. In distributed systems, a major problem is the leader failure and the relevant leader election algorithm. The election algorithms were varied based on the following:

- The nature of the algorithms (Dynamic vs. Static) ([6], [11], [21], and [22]).

- Node Identity (ID) (unique identity vs. anonymous ID) (Distinguished vs not distinguished) [33].
- Topology Type (ring, tree, complete graph, meshes, torus, hypercube …etc) ([1], [8], [21], and [22]).
- Communication mechanism used (synchronous vs. asynchronous) ([21], [22]).
- Transmission media (wired vs. wireless or radio) [12].
- Some of the previous work dealt with the link failure ([1], [25]).

The leader election solution was first thought of at the end of the seventies, it was started by the ring and complete networks ([1], [17], [26]) . In the nineties meshes, hypercube and tree were studied. To date, these topologies and wireless networks are still being studied ([12], [16]).

This section will look over some previous work in election algorithms and focus on the most relevant researches.

In [25], Singh G. proposed a protocol for leader election tolerant to intermittent link failure in the complete graph network. He assumes that up to N/2 – 1 links incident on each node may fail. So, up to $N^2/4 – N/2$ links overall the system may fail. Nodes represent the processors and edges represent bi-directional communication channels between the processors. In leader election problems, all nodes are initially passive. An arbitrary subset of nodes, called the candidate nodes, wake up spontaneously and start the protocol. On the termination of the protocol, exactly one node must announce itself as a leader. The protocol depends on the fact that for any pair(i,j) of nodes, there exists a node k such that both i and j have no faulty link to k . The protocol is composed of iterations. Each iteration is composed of phases. When the iterations reach (log N + 2) the node is the leader. The message complexity of the protocol is $O(N^2)$ [25].

In [17] Molina-G. Presented an algorithm to solve the leader failure for a complete network. When a process notices that the leader is no longer responding to requests, it initiates an election. A process P, holds an election as follows:

1- P sends an election message to every process with the higher number.
2- If no one responds, P wins the election and becomes the leader.
3- If one of the higher numbers answers, it takes over P's job.
4- At any time the old leader recovers it takes over the leadership so this algorithm is called Bully.

In [10] Fredrickson and Lynch, the study assumes the processes are physically or logically ordered, So that each process knows who its successor is. The election message is built when any process notices that the leader is not functioning. The process sends messages containing its number to the successor. If the successor is down, the sender will skip over it and go on to the next number along the ring. During each step the sender adds its own number to the list in the message. Eventually, the message gets back to the process that started it all. That process recognizes this event when it receives an incoming message containing its own process number. At that point, the process sends a leader message to inform all the processes about the new leader.

In [11] Gerard, proposed an election algorithm for oriented hypercube, where each edge is assumed to be labeled with its dimension in the hypercube. When N represents the size of the cube, the algorithm exchanges O(N) messages and uses $O(Log_2N)$time steps to solve the problem in the simple case, when one process detects the leader failure. In more complicated cases when the failure is detected by subset of the processes, the time complexity is linear, and the algorithm terminates in O(N) time steps .

Abu-Amara and Loker, [1] consider the problem of ,fault tolerant, leader election in asynchronous complete (fully connected) distributed networks. They assume that the processors are reliable, but some of communication channels may fail intermittently before or during the execution of the algorithm. Channel failures are undetectable due to asynchronous nature. When N represents the number of processors in the network, and F represents the maximum number of faulty channels on each processor, where F <= (N-1)/2, this algorithm uses at most $O(N^2+NF^2)$ messages to elect a unique leader .

In [8] the election problem in hypercube networks was studied, by using two models with sense of direction, the dimensional and the distance models. The proposed algorithm needs $O(\log^3 N)$ time steps using O(N) messages.

Antonoiu and Srimani [3] a self-stabilizing algorithm for leader election in a tree graph was proposed. Nodes are assigned unique identification

numbers. Each node maintains an ordered list of its neighbors and the predecessor pointer to point to one of its neighbors or null. When the algorithm terminates (in finite time), there is a unique node with a level value that is strictly greater than the levels of all other nodes; this is the elected leader node and each of the rest of the nodes has a unique way to reach that leader. The nodes in the tree are treated uniformly in the sense that each node executes a single uniform rule. Each node has only a partial view of the global state. It knows its own state and the states of its neighbors. Starting from any illegitimate state, the algorithm can elect an arbitrary internal node to be the new leader; but no leaf node will ever be selected as the leader of the tree (a leaf node in a tree is a node with exactly one neighbor).

In [18] Navneet M., Jennifer L., Welch, Nitin and V. presented two new leader election algorithms for mobile ad-hoc networks. The algorithms ensure that eventually each connected component of the topology graph has exactly one leader. The algorithms are based on routing algorithms called TORA. The algorithms require nodes to communicate with only their current neighbors, making it well suited to the ad hoc environment.

In [28] Sudarshan V., Decleene B., Immerman N. , Kurose J.,Towsley D., proposed two cheat-proof election algorithms: Secure Extreme Finding Algorithm (SEFA), and Secure Preference-based Leader Election Algorithm (SPLEA). Both algorithms assume asynchronous distributed system in which the various rounds of election proceed in a lock-step fashion. The SEFA assumes that all elector-nodes share a single common evaluation function that returns the same value at any elector-node when applied to a given candidate-node. When elector-nodes can have different preferences for a candidate-node, the scenario becomes more complicated. The SPLEA deals with this case. Here, individual utility functions at each elector-node determine an elector-node's preference for a given candidate-node.

Most of the previous researchers depended on mathematical proof to verify their algorithms. They used the big O notation to obtain the complexity [15] of the number of messages and time steps, which represent the domain factors of the algorithm complexity ([8], [10]). Other researchers used simulation to validate their algorithms [24].

3 Model Descriptions.

Three-dimensional torus interconnection networks have been used in recent research and commercial distributed memory parallel computers. Examples of such multicomputers are the IBM BlueGene/L [33], the Cray T3D [36], the Cray XT3. An important advantage of the 3D torus over the 2D torus is its lower diameter and higher bisection width, which means that it can achieve reductions in communication delays given the same number of processors.

In 3D Torus network, interconnection topology is a torus graph with N = X * Y * Z nodes (X is the number of nodes in the X dimension, and Y is the number of nodes in the Y dimension, and Z is the number of nodes in the Z dimension of the torus network). This section explains the model description, properties and design assumptions for this research ([31],[32]).

The 3D torus network does is similar to 3D mesh, except in the connection between the first and the last nodes (boundaries) in each dimension. These connections make all nodes connected with six neighbors (Left, Right, Front, Back, Up and Down) to present more flexible topology ([31],[32]). Figure-1 shows three dimensional torus network (7, 4 ,4).

3 Model descriptions:

A d-dimensional hypercube network is represented by N-nodes labeled by d binary bits from (0 to 2^d) (d equivalent to Log N). These nodes are connected by (NlogN)/2 bidirectional links. Node is identifies by binary number composed of d bits called node label. The difference between any two neighbor nodes is only in one bit in the labels. Distance between any two nodes equal the hamming distance between their canonical labels. The diameter and radius of the hypercube equal (Log N).

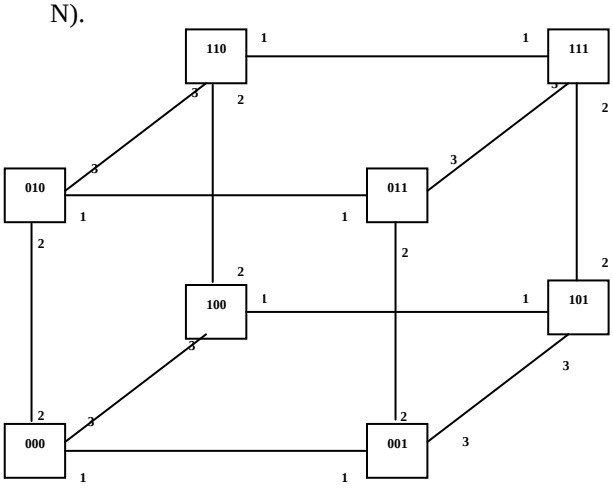

Figure -1 3 Dimensional hypercube

The shortest path between any two nodes is less than or equal (Log N). This path can be found by using Exclusive OR (XOR) operation between the source label and destination label. Hypercube topology has elegant recursive structure. To construct labeled (d+1) dimensional hypercube, we take two d dimensional hypercube and extend all labels in the first dimension with 0 in the left and all labels in the second hypercube with 1. Then, for each node in the first hypercube, add an edge (of direction d) to connect it with associated node in the Second hypercube. Degree of hypercube is equal d. Degree defined by the number of links connect one node by its neighbors. links between nodes are labeled by using nodes labels. Each link connects two nodes derives its label from the order of different bit in two nodes labels. This order is from 1 to d. Figure 1 shows the 3-dimensional hypercube with labeled nodes and links.

This study assumes the following:

- No link failure occurs during algorithm execution.
- All communication links are bidirectional.
- Routers should work all the time even with fault node; because the fault is in leader properties.
- Leader node could fail due to different reasons which will lead to loss of the leadership property. Other nodes can detect this failure when the time out exceeds without acknowledgement. Nodes which detect this failure start the election algorithm.
- To solve leader failure problem, each node calculates a weight number that defines its relative importance. Then, compares it with the weight of other nodes that it has received and propagate the maximum weight. This weight is represented by identification number (ID) for each node. The election algorithm depends on this ID. This research solves the problem when ID number is not distinguished.
- When the leader node crashes, its ID degrades to 0. So, it can not win the election.
- Leader failure may be detected by a subset of nodes (concurrent failure). This case becomes complicated when the failure is detected by N-1 nodes (worst case).
- Each node has the following variables:
  - ID: weight number that defines node relative importance in election process.
  - Position: The label indicates its position.
  - Leader ID , Leader position.
  - Phase and step.
  - State: leader or normal or candidate.

4 Proposed Leader Election algorithms:

Proposed algorithm consists of three phases. Phase one is started, when one or more nodes detect leader failure. After logN time steps, phase one reduces the count of participated nodes in the election process to N/2 nodes aware of the election process. In second phase our algorithm uses the reduction all-to-one communication operation to have the result in one node with address (X10203…0d) (X means 1 or 0). Finally, in the third phase node (X10203…0d) broadcasts the leader message to all nodes in the network. During each step in phases one and two, received ID is compared with the local ID and Greater ID is passed to the next step. Detail description for all phases is as follow:

Phase One starts when one or more nodes detect the leader failure. Each node detects leader failure initiates election process in step one by sending election message to neighbor node that differs in the right most bit. Election message composed of (phase, step, winner ID, and winner position) through link 1.

Step2: the sender and receiver in the previous step send an election messages to the two associated nodes, which differ in the second bit from left, through link 2.

Step3: the senders and receivers from the previous step send an election messages to associated nodes, which differ in the third bit from left, through link 3.

Step Log N: N/2 nodes (the senders and receivers from the Log N –1 step) send an election messages to the nodes, which differ in the Log N bit from left, through link Log N.

During the execution of phase one, if the receiver is aware of the failure and is in progress with its own initiated election step, it will complete the greater step and terminate the smaller one. Each node receives the election message, it compare its own ID with received ID then complete the next step with the greater ID. **If the received ID equal the local ID, algorithm select the ID with greater position to complete with.**

Phase one ends after Log N steps, reducing the participant nodes to N/2. The leader ID and its position for the whole hypercube becoming inside (Log N –1) dimension hypercube.

Phase Two: The second phase uses the reduction all-to-one communication operation to guide the result towards the process that have the address X10…0d. As follows:

Step1: nodes with the second left bit = 1 X1X…Xd send election message to nodes with the second left bit = 0 (X0X…Xd) through link Log N - step.
Step2: the receivers in the previous step with third bits = 1, (XX1…Xd) send election messages to the nodes differ in third bits (XX0..Xd) through LogN - step.
Step (Log N – 1): the receiver in the previous step with Log N bit in its label= 1 (X00…1d) , sends an election message to the process that differ in the right most bit (X00…0d) through log N - step.
After the end of phase 2 the last node (X00…0d) has all information about new leader. This node broadcast leader information in phase 3.
Phase3: In this phase node(X00…0d) broadcasts a message containing the result of the election using one-to-all broadcast operation, the broadcasted message (leader message) contains new leader ID.

5 Abstract Algorithm
This section presents the pseudo code for the election algorithm. Assumptions and variables are assumed here, to use in pseudo code as follow:

- Each node has the following variables:
  a. Local ID: node ID use to participate in election process.
  b. Local Pos: The node Position.
  c. Curr_Step: Last step in the election process.
  d. Ph1_finish_flag: if true it indicates that the Phase 1 was finished.

- The algorithm uses two types of messages:

1. Election message: contains Phase (1, 2 OR 3), step, ID (winner ID), Pos (winner position).
2. Leader message: contains the new Leader position.

- Nodes are in one of following states:
  1. **Normal:** when the node is unaware of any failure and the network is stable.
  2. **Candidate**: when the node is aware of leader failure and participates in election process.
  3. **Leader:** one node must have this state in a stable network.

1. Case state = normal
Upon detect failure
{
State = Candidate
Phase = 1
Step = 1
ID = Local_ID
Pos = Local_Pos
Curr_Step = Step
Send Election(Phase, Step ,ID, Pos) on Link 1.
}
Upon received election message on link r if (Phase == 2)
Store the message and wait until the state becomes candidate and Phase 1 finish. else
{
State = Candidate
if (ID < Local ID)
{
ID = Local ID
Pos = Local Pos
}
if (ID = Local ID)
{
ID = ID of node with greater position
}

if (r < Log N)
{
Step = Step+1
r = r+1
Curr_Step = Step
Send Election(Phase, Step ,ID, Pos) on Link r .
}
if (r = Log N)
{
Ph1_finish_flag = true
if (node label = (XX…1X))
{
Phase = 2
Step = 1
r = Log N – Step
Curr_Step = Step
Send Election(Phase, Step ,ID, Pos) on Link r .
}
}
}
2. Case state = Candidate
Upon Receive Election message
If (Phase = =1)
{
If (Curr_Step > Step)
Ignore the message
If (Curr_Step == Step) and (the r bit in the node label = =1)
Ignore the message
If (Step > Curr_Step) OR ((Curr_Step == Step) and (the r bit in
the node label ==0)) and (r < Log N)
{
Step = Step+1
r = r+1
Curr_Step = Step
Send Election(Phase, Step ,ID, Pos) on Link r .

}
if (Step > Curr_Step) OR ((Curr_Step == Step) and (r == Log
N)) and (node_ label == (XX…1X))
{
Phase = 2
Step = 1
r = Log N – Step
Curr_Step = Step
Send Election (Phase, Step, ID, Pos) on Link r.
}
}
*if (Phase == 2) and (*Ph1_finish_flag = True *)*
{
if (Step< LogN –1)
{
Step = Step +1
r = Log N – Step
Send Election(Phase, Step ,ID, Pos) on Link r
}
if (Step == LogN –1)
BROADCAST LEADER(ID, Pos)
}

6 Performance Evaluations

Proposed algorithm is analyzed by computing number of messages and time steps overall execution of the algorithm. Analyses process is carried out for two cases. Simple case when leader failure is detected by one node, and when leader failure is detected by subset of nodes reach to all nodes in the worst case.

In the following sub-section number of messages and time steps are computed for all cases.

6-1 Number of Messages:

Theorem 1: *Assume that we have N number of nodes in hypercube network. Then, leader election algorithm needs O(N) messages to complete.*

Proof

To find number of messages overall leader election algorithm in the simple case, we compute this number for each phase, then summation of all numbers is calculated.

Phase One:

During this phase each node receives one message except the initiator. So the number of messages is equal (N-1). Another way to compute the messages during phase one is as follow:

Step 1: Needs one message from the initiator to the node that differ in the most right bit

Step 2: Needs two messages from the participated nodes in step1 and so on to log N step. This is shown in formula (1) bellow

$$2^0 + 2^1 + 2^2 + 2^3 + \ldots + 2^{\log(N-1)} = N - 1 \qquad (1)$$

Phase Two: each node sends one message during the second phase (reduction phase)

Except the last node, so the number of messages is equal to **(N/2 – 1)**. By other way we compute the messages during phase as follow: In step1 N/4 election messages are sent, in step2 N/8 is sent until the last step which needs N/N message as shown in formula (2)

$$N/4 + N/8 + \ldots + N/N = (N/2 - 1) \qquad (2)$$

Phase Three: Broadcast needs N-1 messages, since each node receives one leader message except the initiator, as in formula (3)

$$1 + 2 + 4 + 8 + \ldots + N/2 = N - 1 \qquad (3)$$

Total number of messages overall leader election algorithm is given by formula (4)

$$(N-1) + (N/2 - 1) + (N - 1) = 5N/2 - 3 \qquad (4)$$

This result can be expressed in big O notation as follow: O(N) messages which proof theorem 1

Theorem 2 Assume that we have N number of nodes in hypercube network. Then, leader election algorithm needs at most O(N log N) messages to complete.

Proof.

To find number of messages overall leader election algorithm in the worst case, we compute this number for each phase, then summation of all numbers is calculated.

Phase One: Each node sends one message during each step in the first phase. The total is N Log N messages.

Phase Two Three are running in same way as in the simple case. Proof of these phases is covered in theorem 1**.**

Total number of messages for leader election algorithm in hypercube in the worst case is as in formula (6):

$$N(\text{Log } N) + (N/2 - 1) + (N - 1) \qquad (6)$$

When using big O notation the algorithm needs:

O (N log N) messages

6-2 number of Time Steps:

Theorem 3 *Assume that we have N number of nodes in hypercube network. Then, leader election algorithm needs O(Log N) steps to complete in all cases.*

Proof:
Number of time steps is computed for each phase. Then add these numbers to get the total number of time steps overall the algorithm. We apply the computations at the simple case and then at the worst case as follow:

Phase One Reduces nodes to one half of the N that contain the leader ID and position required **Log N time steps** as in the following steps:
Step 1: node detects the failure sends the election message to node that differ in the first right bit.
Step 2: nodes aware of election process send election messages to nodes that differ in the second right bit.
Step Log N: nodes aware of election process send the election messages to nodes that differ in the Log N right bit.

In phase 2 election algorithm continues with d-1 dimensional hypercube. All nodes inside this half are aware of the election process. Nodes in this phase are aware of election results from the first phase. Reduction algorithm is used to guide the result of election to one node in (Log N –1) Steps**.**

In phase three Broadcast leader message use (One-To-All) algorithm hypercube; which needs Log N time steps.
Total steps overall leader election algorithm as in formula 7
Log N + Log N -1 +Log N = 3 Log N – 1 (7)
When use big O notation the algorithm needs :
O (Log N) steps

8. Results and Conclusions
Results in this paper shows that the proposed algorithm presents a distributed leader election algorithm hypercube network. The algorithm solves the problem when ID number is not distinguished. Contention and synchronization issues were considered in designing the algorithm. Performance evaluation is calculated and proofed. For network of N nodes connected as a hypercube network we need only O (N) messages in simple case and O(Nlog N) in the worst case . For both cases the algorithm is completed in O (log N) time steps

References

[1] Abu-Amara, H. and Lokre, J.(1994) **Election in Asynchronous Complete Networks with Intermittent Link Failures,** IEEE Transactions on Computers, Vol. 34 No. 7, July 1994, pp. 778-788.

[2] Akbar B., and Effatparvar Mohammed., and Effatparvar Mahdi, (2006)**, Bully Election Algorithm Improvement with New Methods and Fault Tolerant Mechanism**, Symposium Proceedings Volume II Computer Science & Engineering and Electrical & Electronics Engineering, European University of Lefke, North Cyprus, PP 501-506.

[3] Antonoiu, G. and Srimani, K.**(1996)A Self-Stabilizing Leader Election Algorithm for Tree Graphs,** Journal of Parallel and Distributed Computing, 34, Article No. 0059, 1996, pp. 227-232.

[4] Coulouris G., Dollimore J., and Kindberg T. , (2005), **Distributed Systems Concept and Design**, Fourth Edition, Addison-Wesley, USA

[5] Devillers M., Griffioen D., Romijn J. and Vaandrager F., (2004) , **Verification of Leader Election Protocol, Formal Method Applied to IEEE 1394**, Springer International journal on Software Tools for Tecknology Transfer(STTT), December 2004.

[6] Dolev S., Israeli A. and Moran S., (1997), **Uniform Dynamic Self-Stabilizing Leader Election**, IEEE Transaction on Parallel and Distributed Systems, VOL 8,NO.4, April .PP 424-440.

[7] Duato, J. Yalamanchili, S. and Ni, L. , (1997) **Interconnection Networks an Engineering Approach,** IEEE Computer Society, The Institute of Electronic Engineers, Inc, Los Alamitos, California.

[8] Flocchini, P. and Mans, B. (1996).**Optimal Elections in Labeled Hypercube,** Journal of Parallel and Distributed Computing 33, Article No. 0026, pp. 76-83.

[9] Foster I.(1994).**Designing and Building Parallel Programs**, Addison-Wesley Publishing Company, USA.

[10] Fredrickson, N., and Lynch , N.(1987).**Election a Leader in Asynchronous Ring**, Journal of the ACM, Vol.34, PP. 98-115.

[11] Gerard ,T.,(1993). **Linear Election for Oriented Hypercube**, Technical Report TR-RUU-CS-93-39, Department of computer Science, Utrecht University, The Netherlands.

[12] Jean-Franqois Marckert (2005), **Quasi-Optimal Leader Election Algorithms in Radio Network with Log-Logarithmic Awake Time Slots**, F.chyzak(ed.),INRIA,pp.97-100.

[13] Junguk L. and Geneva G., (1996), **A Distributed Election Protocol for Unreliable Networks,** Journal of Parallel and Distributed Computing, 35, PP 35-42**.**

[14] Kumar V. , Grama A. , Gupta A. and Karypis G. (2003).**Introduction to Parallel Computing**, The Benjamin/Cumminy Publishing Company, Inc, Redwood City, California.

[15] Levitin A**., (**2003), **Introduction to The Design and Analysis of Algorithms,** Addison Wesley Company, USA.

[16] Miroslav K., and Wojciech R., 2004, **Adversary Immune Leader Election in Ad Hoc Radio Networks** [Online]. Available at cs.huji.ac.il/labs/.../adhoc/kutylowski_2003adversdaryimmuneleader.pdf, (verified 2 Mar. 2007).

[17] Molina G, H., (1982).**Elections in A Distributed Computing systems,** IEEE Transactions on Computers, Vol. 31 Jan 1982, pp. 48-59.

[18] Navneet M., Jennifer L., Welch, Nitin V., (2001), **Leader Election Algorithms for Mobile Ad Hoc Networks,** by NSF grant CCR-9972235.

[19] Ostrovsky, R., Rajagoplan, S., and Vazirani, U.,(1994), **Simple and Efficient Leader Election in the Full Information Model**. In Proceedings of the Twenty-Sixth Annual ACM Syposium on Theory of Computing.

[20] Power H.., (1999), **Algorithms and Application in Parallel Computing,** WIT Press/Computational Mechanics Publications, USA.

[21] Refai M. and Ababneh E., (2002) **Leader Election Algorithm in 3D Torus Networks**, Master Theses, not published, Al-Albayet University – Jordan.

[22] Refai, M. and Ajlouni, N**., A new leader Election Algorithm in Hypercube Networks**, Symposium Proceedings Volume II Computer Science & Engineering and Electrical & Electronics Engineering, European University of Lefke, North Cyprus, PP 497-501, 2006.

[23] Richard E. and Kumarss N.,(2004), **Foundations of Algorithms Using Java PseudoCode**, Jones and Bartlett Publishers,Canada.

[24] Russell, A., Saks, M., and Zuckerman, D.,(1999) **Lower Bounds For Leader Election And Collective Coin-Flipping In The Perfect Information Model**. In Proceedings of the Symposium on the Theory of Computing (STOC).

[25] Singh G., (1996). **Leader Election in the Presence of Link Failures**, IEEE Transactions on Parallel and Distributed Systems, VOL 7,No 3,March.

[26] Singh, G., (1991), **Efficient Distributed Algorithms for Leader Election in Complete Networks**, 11th IEEE Int. Conf. on Distributed Computing Systems, PP 472-479.

[27] Singh G., (1997)**, Efficient Leader Election Using Sense of Direction**, Department of Computing and Information Sciences, Kansas State University, Manhatten, KS66506.

[28] Sudarshan V., DeCleene B., Immerman N., Kurose J. and Towsley D. **Leader Election Algorithms for Wireless Ad Hoc Networks**. In Proc. Of IEEE DISCEX III, 2003.

[29] Tanenbaum, A., (2002). **Distributed Systems**, Prentice-Hall International, Inc, New Jersey.

[30 Tanenbaum, A., (1995). **Distributed Operating Systems**, Prentice-Hall International, Inc, New Jersey.

[31] William K., Nellson d., and Ryan S., 2001, **Drawing Graph on the Torus** [Online]. Available at

http://bkocay.cs.umanitoba.ca/g&g/articles/Torus.pdf , (verified 15 Mar. 2007).

[32] William K., and Winnipeg M., 2001, **Embidings of Small Grapg on the Torus** [Online]. Available at: http://bkocay.cs.umanitoba.ca/g&g/articles/Embeddings.pdf, (verified 16 Mar. 2007).

[33] Yamshita M. and Kammeda T.,(1999), **Leader Election Problem on Networks in which Processor Identity Numbers are not Distinct**, IEEE Transactions on Parallel and Distributed Systems, VOL 10,No 9,September.

[34] Abhinav B. and Laxmikant V. Kal ,**Scalable Topology Aware Object Mapping for Large Supercomputers**, PHD Dissertation, Department of Computer Science, University of Illinois at Urbana-Champaign, 2009.

[35] IBM Blue Gene Team. Overview of the IBM Blue Gene/P project. IBM Journal of Research and Development, 52(1/2), 2008.

[36] Kessler, R., and Schwarzmeier, J. "**CRAY T3D: A New Dimension for Cray Research**," Proc. COMPCON, pp.176-182, 1993.

Author:

2. Dr. Mohammed Al Refai *received his PHD in computer science(CS) from Amman Arab University for Graduated studies, Jordan, 2/2007, M.S degree in CS from Alalbayet university, Jordan, 3/2002. he received his undergraduate studies in CS from mutah university, Jordan, 6/1992. He is currently work as lecturer in computer science in KASIT in university ofJordan. His main research interests include many aspects in parallel and distributed systems, Simulation and Data Mining.*