Volume 5, Issue 9, September 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

An Effective Network Utilization Technique in Ad-hoc Sensors for Self-Stabilization Using Fuzzy Approach

T. Madhu[1], SSVN Sarma[2], JVR Murthy[3]
[1]Research Scholar, JNTU Kakinada, A.P., India
[2]Professor, Dept of CSE, Vagdevi College of Engineering, Warangal, Telangana, India
[3]Professor, Dept of CSE, JNTU Kakinada, A.P., India

Abstract: ***The evolution of ad hoc networks fails to self-configure, so it curiously raises fault tolerance issue in which these systems running on such devices has limited configurations, power, and reliability etc., because these nodes are managed autonomously. Generally a fault tolerant distributed algorithm should stabilize itself without any discrepancy in the service. The study illustrates that handling of fault tolerance distributed algorithms is very complex, because the algorithm must support autonomous architecture of network topology and all its possible faults into consideration. The maximum weight based matching problem is a primary problem in graph theory for several autonomous applications. One of conventional issues of clustering theory is the computation of node hops. At present many self-stabilizing methods for this issues undertake clustering approach such as Fuzzy Relevance based Cluster head Selection Algorithm (FRCA), cluster head selection algorithm etc. Although numerous autonomous methods have been proposed to determine node head selection for determining convergence but they do not have stability on ad hoc networks. This work presents a deterministic self-stabilizing algorithm in ad hoc network for anonymous sensor nodes based on fuzzy algorithms. A node can recover from a fault, using the proposed recovery algorithm despite of various autonomous faults present in the an-hoc networks. The proposed algorithm connects the nodes in O(n) steps, where n is the number of nodes.***

Keywords: self-stabilization, ad-hoc networks, faults, graph theory, fuzzy algorithms.

I. INTRODUCTION

A Sensor is a type of distributed computational setting system which supports node sharing, harmonized use of geographically pervasive nodes and multi-owner ability independently from their physical type and location in dynamic virtual organizations that share the same goal of solving large-scale applications. Large applications executing on Sensor or cluster architectures consisting of computational nodes create problems with reliability. The source of the problems is node failures and the need for dynamic configuration over extensive runtime [1, 6 and 9].

Sensor and cluster architectures have gained popularity for computationally intensive parallel applications [5, 7]. However, the complexity of the infrastructure, consisting of sensor nodes, mass storage, and interconnection networks, poses great challenges with respect to overall system reliability. Simple tools of reliability analysis show that as the complexity of the system increases, its reliability, and thus, Fuzzy based clustering algorithm (FCA), decreases. If one models the system as a series reliability block diagram, the reliability of the entire system is computed as the product of the reliabilities of all system components. For applications executing on large clusters or a Sensor, the long execution times may exceed the FCA of the infrastructure and, thus, render the execution infeasible. The high failure probabilities are due to the fact that, in the absence of fault-tolerance mechanisms, the failure of a single node will cause the entire execution to fail. Note that this simple example does not even consider network failures, which are typically more likely than computer failure. Fault tolerance is, thus, a necessity to avoid failure in large applications, such as found in scientific computing, executing on a Sensor, or large cluster.

II. ADHOC SENSOR NETWORKS

In Adhoc Sensor Networks varying node availability becomes dynamic places, often resulting in loss of configuration. To ensure good performance, fault tolerance should be taken into account. Commonly utilized techniques for providing fault tolerance in Sensor Networks are Self-Stabilization. While very robust, these techniques can delay job execution if inappropriate Self-Stabilization times and recovery are chosen. In this thesis introduces several heuristics that dynamically adapt the abovementioned parameters based on information on Sensor status to provide high job throughput in the presence of failure while reducing the system overhead. Simulations (using Perl) are run employing workload and system parameters derived from logs that were collected from several large-scale parallel production systems.

In this context of our problem to have an efficient fault tolerance mechanism, this paper comes up with an optimal Self-Stabilization algorithm that reduces overhead caused due to Self-Stabilization. The proposed system uses Job recovery to ensure completion of work and Dynamic Load balancing is used to avoid overload in any nodes and to achieve maximum node utilization and maximize throughput.

The main objective of Adhoc Sensor Networks is to execute the user messages. Therefore, users submit their messages to the Sensor Scheduler along with their QoS (Quality of service) requirements. These requirements may include the deadline in which users want messages to be transferred, the type of the nodes required to execute the job and the type of the platform needed [11, 12, 16, a8].

To take in hand this predicament, the Fuzzy based clustering algorithm (*FCA*) of the node is taken into consideration when making scheduling decisions using neural network approach. In this session we have designed an optimal Self-Stabilization algorithm based on the Probability density function. It is observed that observed that both existing algorithms still not able to provide an optimal stability times.

III. SELF-STABILIZATION DESIGN

Self-Stabilization is the commonly used technique for fault tolerance and improving system availability. It stores the status of current application state, and then it can be used for restarting the execution in case of failure avoiding the job to start from scratch.

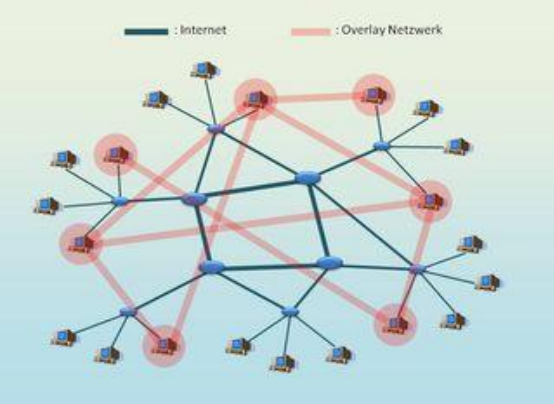

Fig1: Self Stabilization

In the above Fig 1 shows Self Stabilization of application that undergoes during its execution. Once the application has been started, it does some of its work and then pauses to write the stability to stable storage. After completion of stability the application restarts its work [2, 9, 11, 12 and 20].

According to the existing Self-Stabilization scheme, it uses the information about the remaining job execution time, time left before the deadline and the expected remaining needed, to decide whether Self-Stabilization is to be done or not. It also gives information on when the next stability request need to be given. There are two methods in existing Self-Stabilization scheme Fuzzy Relevance Degree (FRD) and Fuzzy Cluster Head Selection Algorithm (FRCA). In FRD algorithm it omits unnecessary stability placement with reference to the total execution time and failure frequency of the node.

IV. SELF-STABILIZATION BASED OPTIMIZED TECHNIQUE (SBOT)

The objective of this work is to optimize the performance of the Sensor in the presence of faults in application. The performance metrics used include throughput, turnaround time and failure tendency. A fault occurs when a Sensor node cannot complete its job within the given deadline. The main strategy of the proposed SBOT depends on using the job Self-Stabilization mechanism to minimize the effect of Sensor faults and to reduce the fault recovery time.

A. Components of the SBOT

The interaction between different components of the SBOT is shown in Fig 2. The SBOT restarts the execution of the failed job from the last saved stability. Thus, it reduces the response time of the job by reducing the time wasted in re-executing partially completed job from the scratch.

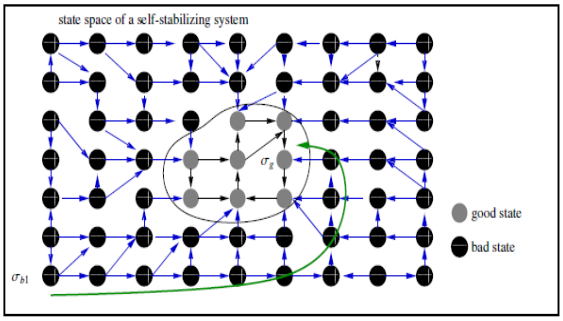

Fig2: Design of the SBOT

A Sensor contains multiple Sensor nodes that provide computing services to users. The main component of the SBOT is the Sensor scheduler (SS). It receives messages with their information from users. Job information includes job

number, job type, and job size. Also, the user submits QoS requirements of each job such as the deadline to complete its execution, the number of required nodes and the type of these nodes.

The main function of SS is to find and sort the most suitable nodes that can execute the job and satisfy user QoS requirements. In order to perform this function, the SS connects to the Sensor information server (SIS) to get information of available Sensor nodes that can execute the job. Fig 2 shows the operation of the SS. The SS uses response time, node failure rate and node failure time to construct the list of suitable nodes that can execute the job.

B. Sensor Scheduling Algorithm

Step 1: Initiate application j messages to Sensor.
Step 2: *for* each job j submitted by the user to the GS;
Step 3: SS gets a list of suitable nodes for j from SIS;

Step 4:SS sorts this list according to the fuzzy centroid $c_j = \frac{\sum_{i=1}^{n} (u_{ij})^m x_i}{\sum_{i=1}^{n} (u_{ij})^m}$ of each node;

Step 5: SS dispatches the message and the list to the SH;

SIS contains information of all available nodes in the Sensor required by the GS. The information includes node speed, current load, node failure rate and total failure time of each node. The latter is the time the node spent in the failure case before it is coming again to join Sensor and work properly.

Stability server (SS) receives and stores partially executed results of a job from the node in times specified by the stability handler (SH). These intermediate results are called stability status. For each job, there is only one record of stability status. When SS receives a new stability status it overwrites the old one. If SS receives a job completion message from the node it removes the record of such job.

SH is an important component of SBOT. The main objective of SH are determining the number of stability nodes, and to determining the stability time for each job. SH receives a job with its assigned list of nodes from GS. It connects to SIS to get information about the failure history of Sensor nodes assigned to the job. Based on failure rate of the node, the SH determines the number of stability's and the stability times for each job. Then, it submits the job to the first Sensor node in the nodes list. The algorithm used by SH to calculate the number of stable nodes and the stability time for each job.

C. C-means clustering algorithm for Self Stability

Let x_i be a vector of values for sensor point g_i.

1. Initialize membership $U^{(0)} = [\, u_{ij} \,]$ for data point g_i of cluster cl_j by random
2. At the *k*-th step, compute the fuzzy centroid $C^{(k)} = [\, c_j \,]$ for $j = 1, .., nc$, where *nc* is the number of clusters, using

$$c_j = \frac{\sum_{i=1}^{n} (u_{ij})^m x_i}{\sum_{i=1}^{n} (u_{ij})^m}$$

Where, *m* is the fuzzy parameter and *n* is the number of data points.

3. Update the fuzzy membership $U^{(k)} = [\, u_{ij} \,]$, using

$$u_{ij} = \frac{\left(\frac{1}{\left\| x_i - c_j \right\|} \right)^{\frac{1}{(m-1)}}}{\sum_{j=1}^{n_c} \left(\frac{1}{\left\| x_i - c_j \right\|} \right)^{\frac{1}{(m-1)}}}$$

4. If $\|U^{(k)} - U^{(k-1)}\| < \varepsilon$, then STOP, else return to step 2.
5. Determine membership cutoff

For each data point g_i, assign g_i to cluster cl_j if u_{ij} of $U^{(k)} > \alpha$

Sensor is a complex environment and the behavior of the nodes in the Sensor is unpredictable. So, it is difficult to build a Sensor on a real scale to validate and evaluate scheduling and fault tolerant systems. Therefore, simulation is often used. However, none of existing simulators support the development of fault-tolerant scheduling algorithms because they have a limited modeling for dynamic Sensors. So, in order to carry out this study, we have used our implemented Sensor simulator.

V. PERFORMANCE EVALUATION

We conducted extensive simulations to evaluate the performance SBOT. The simulations were implemented on TRMSim-WSN - Trust and Reputation Models Simulator for Wireless Sensor Networks [22, 24 and 25], a scalable simulation environment for wireless network systems. The simulator supports modeling and simulation of Sensor nodes and user applications. It enables the creation of messages and mapping of these messages to nodes in the Sensor. In experiments, we modeled applications with size of 100 messages. The size of each job is selected to be 20MB. The number of nodes in the Sensor can reach up to 100. The percentage of faults injected is from 10% to 40%. These specifications remain the same in all experiments of measuring performance.

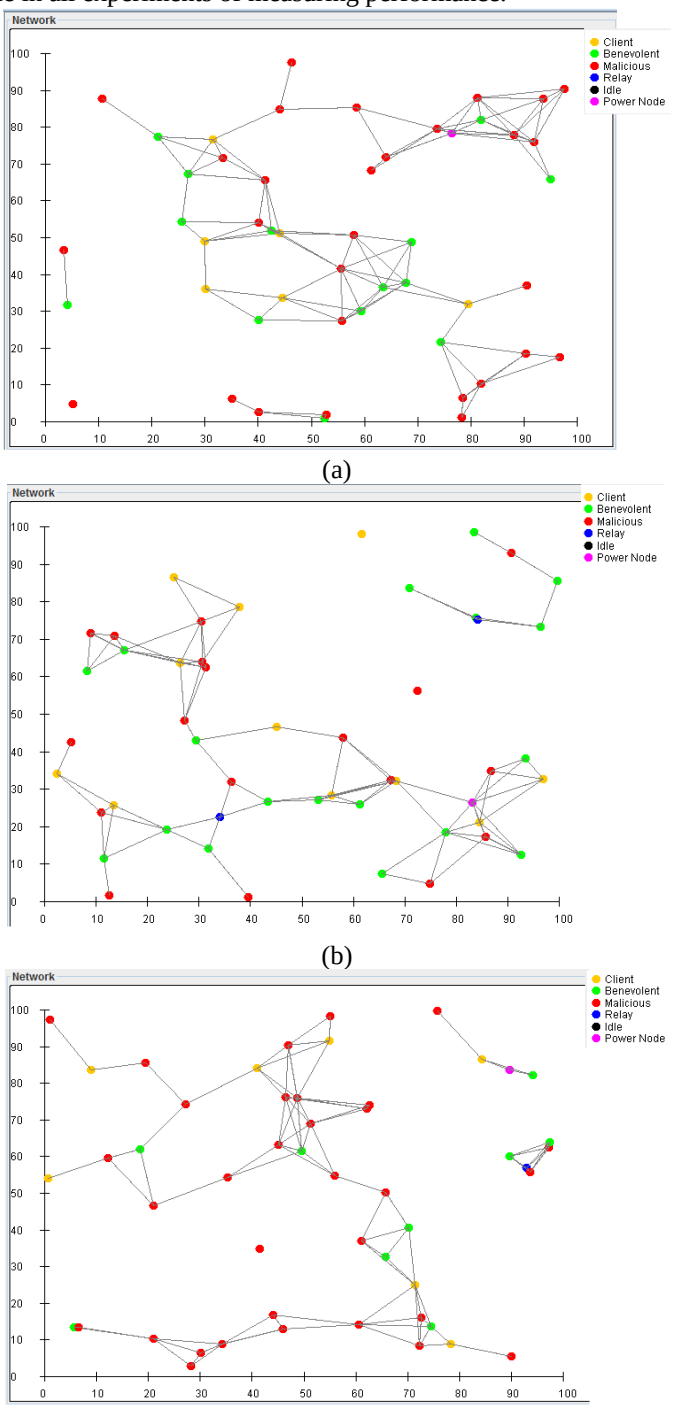

Fig 3: Varying SBOT flows in 100-node networks. (a) Node before simulation. (b) One of Self configurations. (c) Final convergence state.

We have conducted different simulation experiments by varying the total number of faults injected in the Sensor and measuring the throughput, turnaround time and the tendency of nodes to fail. The proposed system is compared with a recent Self-Stabilization-based scheduling system called Stability based Fault Tolerant Sensor System that depends on the response time and node fault index when scheduling messages and uses the fault index of each node when determining the stability times and the number of stability's for each job.

VI. CONCLUSION

In this paper, a Self-Stabilization-based scheduling system for Sensors is proposed and presented. The proposed system depends on Fuzzy based clustering algorithm and failure rate of nodes combined with response time when taking scheduling decisions. The stability time is calculated using node failure rate. The performance of Self-Stabilization Based Optimized Technique (SBOT) is compared with FRCA System scheduling or Existing Algorithm system that depends on the response time and the fault index of nodes when scheduling nodes to transfer messages and SBOT uses the node fault index when calculating stability time. The metrics used for evaluation are throughput, turnaround time and failure tendency.

Experimental results show that SBOT effectively schedules messages in the presence of failures. It is observed that the throughput for the proposed system is better than Existing Algorithm. Also, the SBOT improves the turnaround time when compared with the Existing Algorithm. Moreover, the failure tendency for the proposed SBOT is far better than the Existing Algorithm. Thus, it can be concluded that the proposed scheduling system provides better performance when compared with Existing Algorithm. It implies that consideration of the parameters node failure rate and node failure time improved the performance of SBOT.

REFERENCES

[1] Anish Arora et al. “Project ExScal (Short Abstract)”. In: First IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS). Ed. by Viktor K. Prasanna et al. Vol. 3560. Lecture Notes in Computer Science. Springer, 2005, pp. 393–394 (cit. on p. 1).

[2] Felix C. Gärtner. “Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments”. In: ACM Comput. Surv. 31.1 (1999), pp. 1–26 (cit. on pp. 1, 19, 20).

[3] Edsger W. Dijkstra. “Self-stabilizing Systems in Spite of Distributed Control”. In: Communications of ACM 17.11 (1974), pp. 643–644 (cit. on pp. 1, 10, 18).

[4] Jeffrey O. Kephart and David M. Chess. “The Vision of Autonomic Computing”. In: IEEE Computer 36.1 (2003), pp. 41–50 (cit. on p. 1).

[5] Ian F. Akyildiz et al. “A Survey on Sensor Networks”. In: IEEE Communications Magazine 40.8 (2002), pp. 102–116 (cit. on p. 1).

[6] Fabrice Theoleyre and Fabrice Valois. “About the Self-stabilization of a Virtual Topology for Self-Organization in Ad Hoc Networks”. In: Self-Stabilizing Systems. Ed. by Ted Herman and Sébastien Tixeuil. Lecture Notes in Computer Science 3764. Springer-Verlag, 2005, pp. 214– 228 (cit. on p. 2).

[7] Hongwei Zhang and Anish Arora. “GS3: Scalable Self-Configuration and Self-Healing in Wireless Networks”. In: Proceedings of the twenty-first annual symposium on Principles of Distributed Computing (PODC). ACM, 2002, pp. 58–67 (cit. on p. 2).

[8] Emmanuelle Anceaume et al. “Towards a Theory of Self-Organization”. In: Nineth International Conference on Principles of Distributed Systems (OPODIS). Ed. by James H. Anderson, Giuseppe Prencipe, and Roger Wattenhofer. Vol. 3974. Lecture Notes in Computer Science. Springer, 2005, pp. 191–205 (cit. on p. 2).

[9] Olga Brukman et al. “Self-Stabilization as a Foundation for Autonomic Computing”. In: IEEE ARES 2007 Workshop on Foundation of Fault-tolerance Distributed Computing. 2007 (cit. on p. 2).

[10] Andrew Berns and Sukumar Ghosh. “Dissecting Self-* Properties”. In: Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO). IEEE Computer Society, 2009, pp. 10–19 (cit. on p. 2).

[11] Christel Baier and Joost-Pieter Katoen. Principles of Model Checking. ISBN 0-262-02649-X. The MIT Press, 2008, p. 975 (cit. on p. 2).

[12] Mohamed G. Gouda, Rodney R. Howell, and Louis E. Rosier. “The Instability of Self-Stabilization”. In: Acta Informatica 27.8 (1989), pp. 697–724 (cit. on p. 2).

[13] Abhishek Dhama and Oliver Theel. “A Tranformational Approach for Designing Scheduler-Oblivious Self-stabilizing Algorithms”. In: Proceedings of 12th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS). Ed. by Shlomi Dolev et al. Vol. 6366. Lecture Notes in Computer Science. Springer, 2010, pp. 80–95 (cit. on p. 3).

[14] Avi Silberschatz, Peter Baer Galvin, and Greg Gagne. Operating System Concepts. John Wiley & Sons, Inc., 2008. isbn: 0-470-12872-0 (cit. on p. 5).

[15] Romit Roy Choudhury and Nitin H. Vaidya. “Deafness: A MAC Problem in Ad Hoc Networks when using Directional Antennas”. In: ICNP ’04: Proceedings of the 12th IEEE International Conference on Network Protocols. Washington, DC, USA: IEEE Computer Society, 2004, pp. 283–292. isbn: 0-7695-2161-4 (cit. on p. 5).

[16] Nancy A. Lynch. Distributed Algorithms. Morgan Kaufmann Publishers, Inc., 1996 (cit. on p. 5).

[17] Clyde P. Kruskal, Larry Rudolph, andMarc Snir. “Efficient Synchronization onMultiprocessors with Shared Memory”. In: PODC ’86: Proceedings of the fifth annual ACM symposium on Principles of distributed computing. ACM, 1986, pp. 218–228. isbn: 0-89791-198-9 (cit. on p. 7).

[18] Allan Gottlieb et al. “The NYU Ultracomputer - Designing an MIMD Shared Memory Parallel Computer”. In: IEEE Transactions on Computers 32.2 (1983), pp. 175–189 (cit. on p. 7).

[19] Maurice Herlihy. "Wait-free synchronization". In: ACM Trans. Program. Lang. Syst. 13.1(1991), pp. 124–149. issn: 0164-0925 (cit. on p. 7).

[20] Ambuj K. Singh, James H. Anderson, and Mohamed G.Gouda. "The elusive atomic register". In: J. ACM 41.2 (1994), pp. 311–339. issn: 0004-5411 (cit. on p. 7).

[21] Eli Upfal and Avi Wigderson. "How to share memory in a distributed system". In: Journal of the ACM 34.1 (1987), pp. 116–127 (cit. on p. 7).

[22] Tim J. Harris. "A survey of PRAM simulation techniques". In: ACM Computing Surveys 26.2 (1994), pp. 187–206 (cit. on p. 7).

[23] Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. "Sharing memory robustly in messagepassing systems". In: Journal of ACM 42.1 (1995), pp. 124–142. issn: 0004-5411 (cit. on pp. 7, 8).

[24] Edsger W. Dijkstra. "Guarded Commands, Nondeterminancy, and Formal Derivation of Programs." In: Communications of ACM 18 (1975), pp. 453–457 (cit. on pp. 8, 9).

[25] Nechama Allenberg-Navony, Alon Itai, and Shlomo Moran. "Average and randomized complexity of distributed problems". In: Distributed Algorithms. Vol. 857. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1994, pp. 311–325 (cit. on p. 10).

