NPS-CS-06-013

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

Requirements for Self-Stabilization of Distributed Advanced Battle Managers

by

Mikhail Auguston, Thomas S. Cook, James B. Michael,
Man-Tak Shing, Harsha Tummala,
Duminda Wijesekera, Geoffrey G. Xie

15 September 2006

Approved for public release; distribution is unlimited.

Prepared for: Missile Defense Agency
7100 Defense Pentagon
Washington, DC 20301-7100

THIS PAGE IS INTENTIONALLY LEFT BLANK

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

COL David Smarsh, USAF
Acting President

Leonard A. Ferrari
Acting Provost

This report was prepared for and funded by the Missile Defense Agency.

This report was prepared by:

_____
James Bret Michael, Associate Professor
Department of Computer Science

Reviewed by:

_____
Peter J. Denning, Chairman and Professor
Department of Computer Science

Released by:

_____
Leonard A. Ferrari
Acting Provost and
Dean of Research

THIS PAGE IS INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** 09/15/06	**3. REPORT TYPE AND DATES COVERED** Technical Report

4. TITLE AND SUBTITLE:
Requirements for Self-Stabilization of Distributed Advanced Battle Managers

5. FUNDING NUMBERS
MD6080101P1916

6. AUTHOR(S)
Mikhail Auguston, Thomas S. Cook, James B. Michael, Man-Tak Shing, Harsha Tummala, Duminda Wijesekera, Geoffrey G. Xie

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER
NPS-CS-06-013

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Missile Defense Agency, 7100 Defense Pentagon, Washington, DC 20301-7100

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this technical report are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(maximum 200 words)*** In this report, we formalize the self-stabilization problem as it pertains to the C2BMC, in addition to highlighting some of key features of the C2BMC that distinguish it from general-purpose distributed systems. We then describe a sub-area of self-stabilization known as the leader election problem, pointing out the issues tied to the re-establishment of an ABM command and control structure in the event of system faults. We performed an initial survey of a small sample of recent articles from the open literature on leader election algorithms; we found that there are over 10,000 articles on the leader election problem. The report includes a critique of each of the surveyed articles. Our report concludes with recommendations for both a framework of a leader election protocol that could be applied to the C2BMC and specific directions of further research to be conducted.

14. SUBJECT TERMS
Ballistic Missile Defense, Distributed computing, Failure detection, Self-stabilization, Leader election

15. NUMBER OF PAGES
34

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UL

Standard Form 298 (Rev.2-89)

THIS PAGE IS INTENTIONALLY LEFT BLANK

Introduction

Many challenges exist in the design and implementation of reliable applications on top of asynchronous, distributed systems prone to process crashes. Whenever the application departs from its correct state due to a process crash, the live processes must execute some algorithms to restore the application back to the correct state.

Asynchronous, distributed applications often use a dedicated node's to solve synchronization and coordination problems in order to improve the overall efficiency of the system. The leader election problem deals with the problem of having many nodes compete to become the dedicated node that performs these special tasks [27]. In the context of the Command, Control, Battle Management, Communication (C2BMC) element of the Ballistic Missile Defense System (BMDS), the Global Integrated Fire Control (GIFC) component's Advanced Battle Manager (ABM) leader nodes will coordinate and allocate resources.[1] To do this efficiently the C2BMC requires a strong hierarchical command and control structure for crucial decision making.

In an effort to evaluate leader election algorithms for use in the C2BMC domain, our research group explored a number of papers related to leader election algorithms. It should be noted that "there are over 10,000 articles on the leader election problem" [27]. Our initial survey looks at a small, relatively recent sample of this literature that shows some of the directions that research on leader-election algorithms has taken. We look at the leader-election problem as a specific subclass of the self-stabilization problem. Following this, we describe the relationship of the leader election in the context of the C2BMC and specifically the issues tied to the re-establishment of an ABM command and control structure in the event of a transient fault. We review a few specific features of the C2BMC that make it different from general-purpose distributed systems. We then summarize individual papers on the leader election protocol (See Appendix). After each summary, we discuss how the concepts of the paper might be applied to the missile defense domain. Finally, based on ideas inspired by these papers, we provide a discussion for the leader election protocol that might be applied to the C2BMC and identify the direction of further research.

It is important to define a few terms that are typically associated with distributed systems.

Formalization of the Self-Stabilization problem

Dijkstra introduced the concept of self-stabilizing systems over twenty years ago [9]. He showed through the development of an algorithm that a system that both experiences transient faults and enters an illegal state could self-recover in some finite number of steps, bringing itself back into a legal state to continue its intended mission. Dijkstra's formal definition of self-stabilizing is as follows:

[1] We use the term resource here to mean anything at the disposal of the C2BMC for protecting assets.

> We call the system self-stabilizing if and only if regardless of the initial state and regardless of the privilege selected for the next move, always at least one privilege will be present and the system is guaranteed to find itself in a legitimate state after a finite number of moves.
>
> For a long time it has been an open question whether non-trivial self-stabilizing systems could exist. It is not directly obvious whether local moves can assure convergence towards satisfaction of such a global criterion; the non-determinacy of the privilege selected is an added complication.

Twenty plus years later, we see the self-stabilization property being used in routing algorithms that support large, distributed, heterogeneous communication networks, in control systems for the most recent space shuttles, and in many resource-allocation applications. In today's world, systems without self-stabilizing algorithms are simply unacceptable: it is too costly, in terms of time, money, and in some cases even life, not to have a self-stabilizing system in an operational setting.

Self-stabilization protocol

In the context of the C2BMC we define the predicates for legal and illegal states. We then propose that for the system to self-stabilize it abide by the following logic:

From the perspective of the C2BMC the self-stabilization criteria is as follows: if the system enters an illegal state it will eventually, within some maximum time, t_{max}, enter a legal state. A specific example might be as follows: assume the criteria for a legal state that there exists only a single leader within any partition of the BMDS network. If it were the case that more than one leader or no leader existed in a single partition of the BMDS network, the system would be in an illegal state. The self-stabilization protocol would then, within t_{max}, identify a single leader in each partition, bringing the system back into a legal state. (N.B: The legal state predicates must be computable. That is, there is an algorithm that computes the legal state).

Self-stabilization algorithms are a general class of algorithms that can be broken down further into subclasses. These subclasses deal with specific challenges of stabilization.

One such subclass provides the leader election protocols. The leader election problem is widely known and several thousand algorithms are known in the field of distributed computing [27]. The major concern in defining the legal states for C2BMC is maintaining the hierarchical chain-of-command (CoC). In this context the leader election problem becomes the problem of establishing the correct CoC for dynamically evolving BMDS network configurations.

Challenges in asynchronous distributed systems

The design and implementation of reliable applications on top of asynchronous distributed systems that are prone to processor crashes is a difficult and complex task.

Whenever the application departs from its legal state due to processor crashes, the live processors must execute some protocols to restore the application back to the legal state.

A main issue lies in the impossibility of correctly detecting crashes in the presence of asynchrony [9]. It is impossible to safely distinguish a crashed processor from a processor that is slow or with which communication is slow, intermittent, or becomes unavailable.

To address this issue for C2BMC it is reasonable to assume that there exists a minimum time-out for receiving expected messages. This assumption is crucial for system self-stabilization algorithm. If the concurrent configuration (partitioning) of the C2BMC Network is known then the CoC is straight forward (Standard Operating Procedure). Therefore the major issue of the C2BMC self-stabilization becomes a problem of instant polling for the C2BMC Network configuration. The assumption of the pre-defined minimum time-out is at the core of the network polling process. The network requirement is to keep all ABM nodes connected so that all correct ABM nodes know the current network configuration. In the literature there are known algorithms for determining network configurations within $O(\log n)$ steps where n is the number of nodes [8, 11, 18, 21].

However, the current network control protocols cannot ensure robust connectivity between ABM nodes. Networks controlled by these protocols exhibit a defining characteristic of unstable complex systems—a small local event (e.g., incorrect configuration of a routing protocol on a single interface or a link failure) can have severe, global impact in the form of a cascading meltdown; the meltdown would be manifested by the loss of connectivity between most ABMs in the case of C2BMC. The networking community has begun to realize the severity of this problem and several new approaches to network control and management have been proposed in the literature [15].

It is impossible to build a stable C2BMC system on top of an unstable network. Although the focus of this report is about self-stabilization and leader election, the issue of network instability and different solution approaches must also be investigated in order to obtain valid assumptions about the performance characteristics of the communication channels between ABMs. Since C2BMC will use a dedicated private data network, it is feasible to thoroughly evaluate different types of control protocols for the network, to include prototype implementation.

Leader Election Algorithms for Asynchronous Distributed Systems

Leader oracle

Leader-based protocols rest on a primitive that if run on any processor in the distributed system, outputs the same answer, that we call the leader. Such protocols are common in distributed computing to solve synchronization or coordination problems. A facility essential to these protocols is the leader oracle, which satisfies the following properties:

Property 1 (Eventual Leadership) *A leader is eventually elected, but there is no knowledge on when this common leader is elected and, before this occurs, several distinct leaders (possibly conflicting) can co-exist.*

Unfortunately, guaranteeing such a property is nontrivial for asynchronous distributed systems prone to node crashes, and is impossible (that is non-computable) in fault-prone purely asynchronous systems [9]. To circumvent this difficulty, several protocols have been proposed that build a leader facility on top of an asynchronous distributed system enriched with additional assumptions. For example, a protocol may include one or more of the following assumptions:

Assumption 1 (Known Maximum Outage) *At most f out of n processors can crash.*

Assumption 2 (Reliable Channels) *Channels are reliable.*

Assumption 3 (Eventually Timely Channels) *Let p_i and p_j be two processes. We say that a directed channel p_i -> p_j is eventually timely if there is a time t after which either p_j has crashed or there is a bound $\delta>0$ such that each message sent by p_i after t is received by p_j within δ units of time*. [22]

Assumption 4 (Eventually Winning Channels, Time-Free) *A directed channel $p_i \rightarrow p_j$ from processor p_i to processor p_j is eventually winning if whenever a processor p_j broadcasts a query to the network, there is a finite time t after which p_i's response is always among the first (N-f) responses received by p_j.*

For example, [22] presented an eventual leader election protocol that benefits form the best of time-based and time-free assumptions.

Domain-Specific Features of the C2BMC

The distributed system formed by the replicated ABMs of the C2BMC has specific features that distinguish it from general purpose distributed systems. In the context of the C2BMC a leader among ABM nodes is analogous to the leader in a military CoC; the leader provides orders down the chain (based on guidance from those higher up the chain) and provides situational awareness up the CoC (based on the details provided by subordinates). In the C2BMC it is imperative for ABM nodes at the top of the hierarchy to get *quality* information from its subordinates as this information is used to appropriately allocate resources. It is equally important for ABM nodes at the bottom of the hierarchy to get quality information from its superiors as this information contains execution orders. The following is a non-exhaustive list of some of the specific features that make it unique from general purpose distributed systems:

- The structure of the network of distributed ABM nodes[2] is very specific: There is a small number (~200) of nodes that can execute a known number of commands and with a known communication protocol.
- There is a hierarchy of subordination (i.e., the CoC) among nodes.
- The nodes are statically specified and with hard real-time execution guarantees.
- Messages channels may be subjected to loss, alteration or injection of spurious messages.
- The concept of Area of Responsibility (AOR) is essential for the functionality of ABM's. This means that significant parts of functionality and resources are delegated to the ABM nodes according to the hierarchical CoC. This implies a specific hierarchical pattern of communication
- Some sensors originate messages that convey true information.
- CoC for a particular network configuration depends on resource constraints, geographical location and weather conditions for a node, operation plan (OPLAN), etc.
- It will run over a dedicated private communication network which can be engineered to provide more stringent performance guarantees than the public data networks.

Definition 1. ABM System

We define an ABM system as an ordered pair (N, $\boldsymbol{\ell}$) where

- N is a set of ABM nodes, and
- $\boldsymbol{\ell} \subset$ N x N is a symmetric relation defining the connectivity structure of N.

Definition 2. Live AMB System

Let ***live***(N) $\subseteq$ N and ***live***($\boldsymbol{\ell}$) $\subseteq \boldsymbol{\ell}$ satisfy the condition that, for any two ABM node x_i, x_j, (x_i, x_j) $\in$ ***live***($\boldsymbol{\ell}$) $\Rightarrow$ $x_i \in$ ***live***(N), $x_i \in$ ***live***(N) and there is stable two-way communication between x_i and x_j.

We say that (***live***(N), ***live***($\boldsymbol{\ell}$)) is the *lively fragment* of the ABM system.

Definition 3. Live partitions of an ABM system

Let $p_1, p_2, \ldots, p_m \subseteq$ (***live***(N), ***live***($\boldsymbol{\ell}$)) such that for any two ABM node $x \in p_i$ and $y \in p_j$, $i \neq j$, $(x, y) \notin$ ***live***($\boldsymbol{\ell}$).

We say that P = $\{p_1, p_2, \ldots, p_m\}$ is a *live partition* of (N, $\boldsymbol{\ell}$) if $p_1 \cup p_2 \cup \ldots \cup p_m =$ (***live***(N), ***live***($\boldsymbol{\ell}$)) and $p_i \cap p_j = \varnothing$ for $1 \leq i, j \leq m$. The subsets $p_1, p_2, \ldots, p_m$ will be referred to as the *connected components* of *P*.

[2] ABM nodes consist of one or more processors. An ABM node configuration can be, for example, a single processor, a single processor with auxiliary processors (e.g., hot-swappable), or a mesh of processors.

Definition 4. Dynamic CoC hierarchy

Let $P = \{p_1, p_2, \ldots, p_m\}$ be a live partition of (N, ℓ). We use ***CoC***(P) to denote the chain-of-command hierarchy for the partition P. Note that ***CoC***(P) depends on the particular network configuration induced by P, resource constraints, geographical location and weather conditions for a node, OPLAN, etc.

For any two ABM nodes x, y in a connected component p_i, we say $x > y$ if y is under x according to ***CoC***(P)

Definition 5. Component leader

Given a live partition $P = \{p_1, p_2, \ldots, p_m\}$ with a chain-of-command hierarchy ***CoC***(P), an ABM node x is a *leader* of component p_i if $x > y$ for all ABM nodes y ($x \neq y$) in the component p_i.

Definition 6. Legal Live Partition of an ABM System

Given a live partition $P = \{p_1, p_2, \ldots, p_m\}$ with a chain-of-command hierarchy ***CoC***(P), there is exactly one leader in each component p_i, for $1 \leq i \leq m$.

Thus, according to Definition 1 an ABM system is modeled as an undirected connected graph with a hierarchical partial relation >. A live ABM system is a subsystem where all communication links and nodes are alive. A leader of a live system is a maximal element of a maximally connected live subsystem. The difference between this definition and the normal distributed system is the absence of the hierarchical relation > in the latter and the fact that the leader has to be a highest among the CoC hierarchy >.

For example, consider the ABM system with seven nodes shown in Figure 1. In the beginning, all nodes and links are alive and let P denote the initial partition containing the whole network as one component with ***CoC***(P) denoted in Figure 2.

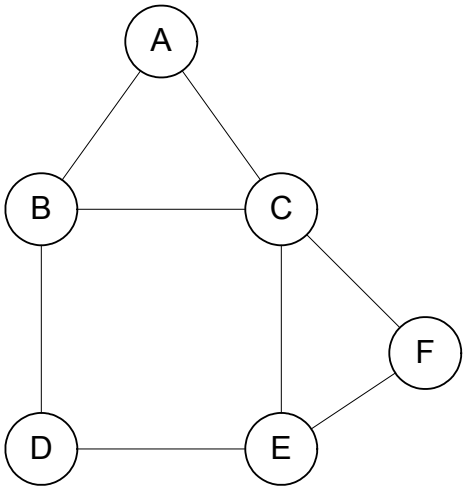

Figure 1

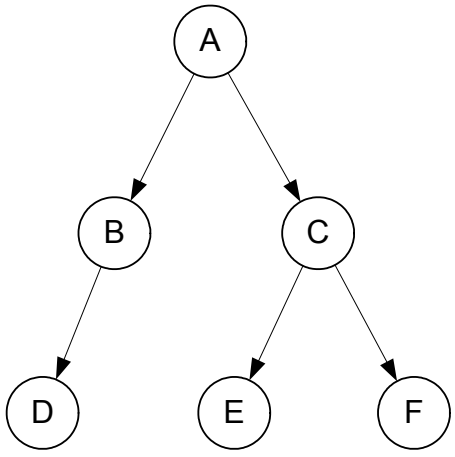

Figure 2

Suppose node *A* and the links (*B*, *D*), (*C*, *E*), and (*C*, *F*) are all gone, resulting in the new live partition *Q* consisting of two components {*B*, *C*} and {*D*, *E*, *F*} (Figure 3). Then, depending on resource constraints, geographical location and weather conditions for a node, OPLAN, etc., the ABM system may switch to ***CoC***(*Q*) shown in Figure 4. Note that *E* > D according to ***CoC***(*Q*), although > is undefined between the nodes *D*, *E* and *F* according to ***CoC***(*P*).

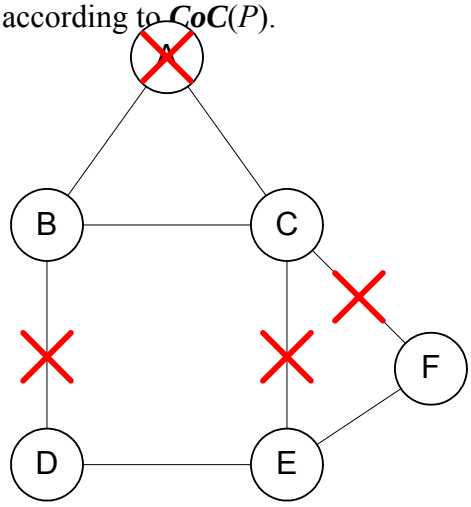

Figure 3

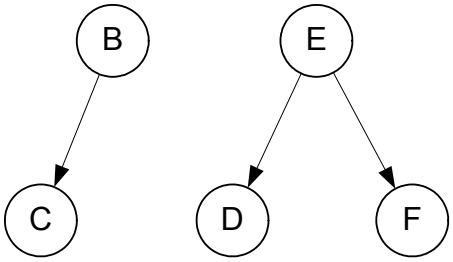

Figure 4

We intentionally define the CoC hierarchy to be as dynamic as possible, providing for maximum flexibility, survivability, and robustness of the ABM system. Here we assume that there is enough time for those ABM nodes that are alive to discover the changing partitions and switch to a new CoC based on a pre-defined OPLAN.

Summary and Discussion of Existing Literature

The appendix contains a short summary of each article and a discussion of the relevance of the content of the article to the C2BMC problem domain.

Discussion and Recommendations

As noted earlier, there are over 10,000 known articles on the leader election problem. From our analysis of a selection of these papers, it is highly improbable that we can directly apply any of these existing solutions to our specific domain. However, it has been shown in [27] how leader election algorithms can be adapted to solve specialized problems similar to the C2BMC. Furthermore, our exploration of existing work has inspired the following approach and recommendations.

We describe how we envision the behavior of the self-stabilizing system depending on the available communication channels, and the determination of leadership within a

known configuration of nodes. In this context, and ignoring other potential problems (see below) the solution is straightforward. For a given configuration of nodes the choice of the leader is unambiguous, and can be predefined. Hence, the solution is based in keeping instant polling of the network in order to be aware about the current node configuration; we recommend this to be one of the core tasks performed by the ABM. The key challenges here are to determine the optimal polling rate and what to do if the current leader is unreachable due to either incorrect processes (e.g., node crashes) or unavailability of one or more channels. We need failure detectors in the form of distributed algorithms [14] to quickly discover the new network partitions and re-establish the CoC in each partition. Algorithms for resource discovery in distributed systems are well known and provide time estimates of approximately $O(\log n)$ for a system with n nodes. While $O(\log n)$ algorithms are probably adequate for all practical purposes, we need to perform an empirical study via simulation to find out if an $O(\log n)$ algorithm can produce the correct results under the appropriate hard real-time constraints. In addition, we need to address another potential problem about the behavior of the system between the configuration transitions; that is, we need to address the consistency of behavior at the transition period taking into account the potential conflict between AOR rules and strictly hierarchical CoC rules.

A possible approach would be to dynamically create a tree of leaders (such as is done, for instance, in distributed-queue dual-bus network (DQDB) algorithms) and dynamically update the tree to reflect the changes to the environment, instead of electing a leader. If we had the best estimate of the next leader on a distributed tree, then that leader can be called upon to perform. We may be able to customize the so-called distributed-space partitioning tree algorithm referred to in the thesis by Heutelbeck [16]. Three advantages to the aforementioned approach are:

(1) Can reflect the military hierarchy
(2) More general than having one leader and no successor
(3) Can have an offline periodic task constantly update the tree so that the dynamic re-computation due to leader's death can be minimized

We must also consider the behavior of the system when some messages passed through existing channels may be untrustworthy. Well-known methods for networks with unreliable communication channels should be examined. The expected solution may require an appropriate level of communication redundancy. We recommend that several trust models be considered for use and that a performance evaluation be performed to determine the timeliness associated with each trust model. The reason for suggesting different trust models is that the trust model depends upon the misuse one has in mind. Potentially, we can use a protocol similar to OTAR (over the air key re-keying) [27, 28, 29, 30].

Another independent problem is resource management within the configuration, which may be addressed by applying well known methods for dynamic programming and optimization.

In addition, the possible set of reactions to a given set of events of an ABM can be predetermined, of which only one action can be taken at a time. However, the time limit within which a reaction is expected will be governed by a (software) time limit. Exceeding the time limit could reduce the effectiveness or performance of the system. One could associate a quality-of-service (QoS) value with the deviation from measures of effectiveness and performance. Conversely, inter-process messages could be subjected to stochastic delays and drops. Therefore, we recommend that the ABMs be modeled as probabilistic temporal automatons.

References

[1] Antonakopoulos, S. Fast leader-election protocols with bounded cheaters' edge. In *Proc. 38th ACM Symposium on Theory of Computing*, ACM (Seattle, Wash., May 2006), pp. 187-196.

[2] Barrière, L., Flocchini, P., Fraigniaud, P., and Santoro, N. Can we elect if we cannot compare? In *Proc. 15th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, ACM (San Diego, Calif., June 2003), pp. 324-332.

[3] Beauquier, J., Gradinariu, M., and Johnen, C. Memory space requirements for self-stabilizing leader election protocols. In *Proc. 18th ACM Symposium on Principles of Distributed Computing*, ACM (Atlanta, Ga., May 1999), pp. 199-207.

[4] Bellettini, C. and Capra, L. Quotient graphs for the analysis of asymmetric distributed systems: surveying two alternative approaches. In *Proc. 19th Int. Parallel and Distributed Processing Symposium*, IEEE (Denver, Colo., Apr. 2005).

[5] Birman, K. P. *Reliable Distributed Systems*. New York: Springer, 2005.

[6] Boichat, R., Frølund, S., and Guerraoui, R. Deconstructing Paxos. *ACM SIGACT News* 34, 1 (Mar. 2003), pp. 47-67.

[7] Chandra, T.D. and Toueg, S. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM* 43, 2 (Mar. 1996), pp. 225–267.

[8] Cidon, I., Gopal, I., and Kutten, S. New models and algorithms for future networks. *IEEE Trans. Information Theory* 41, 3 (May 1995), pp. 769-780.

[9] Dijkstra, Edsger, W., Self-stabilizing systems with distributed control, unpublished manuscript, November 1973.

[10] Dolev, S. *Self-Stabilization*. Cambridge, Mass.: MIT Press, 2000.

[11] Duarte, E. P., Jr. and Weber, A. A distributed network connectivity algorithm. In *Proc. 6th Int. Sympositum on Autonomous Decentralized Systems*, IEEE (Pisa, Italy, Apr. 2003), pp. 285-292.

[12] Fernández, A., Jiménez, E., and Raynal, M. Eventual leader election with weak assumptions on initial knowledge, communication reliability, and synchrony. In *Proc. Int. Conf. on Dependable Systems and Networks*, IEEE (Philadelphia, Penn., June 2006), pp. 166-178.

[13] Fetzer, C. and Raynal, M. Elastic vector time. In *Proc. 23rd Int. Conf. on Distributed Computing Systems*, IEEE (Providence, R.I., 2003), pp. 284-291.

[14] Garg, V.J. *Elements of Distributed Computing*. New York: John Wiley & Sons, Inc., 2002.

[15] Greenberg A., Hjalmtysson G., Maltz, D., Meyers A., Rexford J., Xie, G., Yan, H., Zhan, J., and Zhang, H., A clean slate 4D approach to network control and management. *ACM SIGCOMM Computer Communications Review* 35, 5 (Oct. 2005), pp. 41-54.

[16] Heutelbeck, D. Distributed Space Partitioning Trees and their Application in Mobile Computing. PhD thesis, Fern Universität in Hagen, 2005.

[17] Heutelbeck, D. and Hemmje, M. Distributed leader election in P2P systems for dynamic sets. In *Proc. 7th Int. Conf. on Mobile Data Management*, IEEE (Nara, Japan, May 2006).

[18] Keidar, I. and Shraer, A. Timelieness, failure-detectors, and consensus performance. In *Proc. 25th ACM Symposium on Principles of Distributed Computing*, ACM (Denver, Colo., July 2006), pp. 169-178.
[19] King, V., Saia, J., Sanwalani, V., and Vee, E. Scalable leader election. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, ACM, (Miami, Fla., Jan. 2006), pp. 990-999.
[20] Kutten, S., Peleg, D., and Vishkin, U. Deterministic resource discovery in distributed networks. In *Proc. 13th Annual ACM Symposium on Parallel Algorithms and Architectures*, ACM (Crete, Greece, July 2001), pp. 77-83.
[21] Larrea, M., Fernandez, A., and Arevalo, S. Eventually consistent failure detectors. In *Proc. 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, IEEE (Canary Islands., Jan. 2002), pp. 91-98.
[22] Mostefaoui, A., Raynal, M., and Travers, C. Time-free and timer-based assumptions can be combined to obtain eventual leadership. *IEEE Trans. Parallel and Distributed Systems* 17, 7 (July 2006), pp. 656-666.
[23] Rangarajan, S., Dahbura, A. T., and Ziegler, E. A. A distributed system-level diagnosis algorithm for arbitrary network topologies. *IEEE Trans. Computers* 44, 2 (Feb. 1995), pp. 312-334.
[24] Raz, D., Shavitt, Y., and Zhang, L. Distributed council election. *IEEE/ACM Trans. Networking* 12, 3 (June 2004), pp. 483-492.
[25] Raynal, M. A short introduction to failure detectors for asynchronous distributed systems. *ACM SIGACT News* 36, 1 (Mar. 2005), pp. 53-70.
[26] Shi, W. and Corriveau, J.-P. An executable model for a family of election algorithms. In *Proc. 18th Int. Parallel and Distributed Processing Symposium*, IEEE (Santa Fe, N.M., Apr. 2004), pp. 178-185.
[27] Svensson, H. and Arts, T. A new leader election implementation. In *Proc. ACM SIGPLAN Workshop on Erlang*, ACM (Tallinn, Estonia, Sept. 2005), pp. 35-39.
[28] Villadangos, J., Córdoba, A, Fariña, F, and Prieto, M. Efficient leader election in complete networks. In *Proc. 13th Euromicro Conf. on Parallel, Distributed and Network-based Processing*, IEEE (Lugano, Switz., Feb. 2005), pp. 136-143.
[29] TIA/EIA/TSB-102.AACA, Over the Air Rekeying Protocol, Telecommunications Industry Association.
[30] TIA/EIA/TSB-102.AACA-1, OTA Protocol, Telecommunications Industry Association.
[31] TIA/EIA-TSB-102.AACB, OTAR Operational Description, Telecommunications Industry Association.
[32] TIA/EIA-TSB.AACC-2, Conformance Test for the P-25 OTAR Protocol, Telecommunications Industry Association.

Appendix: Literature Review

Reviewed Article: Distributed Leader Election in P2P Systems for Dynamic Sets [17]

Summary

This paper focuses on constructing a distributed leader election algorithm suitable for P2P systems. The novelty is that the proposed method constructs a distributed space partitioning tree to answer location specific information, including the election of a leader. The two algorithms included in the paper are improvements over those contained in one of the author's doctoral thesis from 2005.

Discussion

The Problem formulation:

There is a geographical region (say) S that is referred to as the context space. That is taken to be an n-dimensional interval. Intervals of this set S are referred to as locations, where each location consists of a compact measurable region of space, and taken together they do not intersect. [The compactness and the measurability are used to ensure that probabilistic laws can be applied to the sub-spaces]. Then q (space related) query (say) Q is broadcasted over S, and individual entities that are specific locations that intersect (the region referred to in) Q are invited to answer (and submit objects O that satisfy Q) Then, more than one peer belonging to a location (say) l can respond to the query Q, resulting in the requestor receiving redundant queries; this is referred to as the *redundant query problem*. The main objective of the paper is to avoid this. The paper presents two algorithms that satisfy the following criteria:

1. All peers in one cluster selects one peer among them to answer the query.
2. The selection of the peer should not cause additional traffic.
3. The selection should be fair and scalable. That is, the probability that the peer of cluster C is selected is approximately $|A \cap C|/|A|$ where $|X|$ refers to the Lebesgue measure of region X (in naïve geographic terms, the area of region X).
4. When processing a query, the selection of a peer should be independent of each matching object.

Algorithm 1: The first algorithm assumes that the locations sets are n-dimensional cuboids aligned with the coordinate axes.
Positive aspects: Very compact representation
Negative aspects: Unrealistic in terms of real-life geometries.
Procedure: The peer sending the query randomly selects a point $p \in [0,1]^n$ and The peers finding a matching located object (l_o, o) can calculated $A := l_o \cap q$. Use an affine transformation to map $[0,1]^n$ to A where p is mapped to p'. Only the peers of all clusters contain candidate points such as p' answers back.

Problem: Requirements (1) and (3), and (2) in some sense but not (4) is satisfied. A small variation is suggested to address this deficiency. In order to avoid having more than one peer answer back, they propose using a **SHA-1** hash-based selection procedure for a peer to determine if it is required to answer the query.

Algorithm 2: The second algorithm relaxes the assumption of having statically assigned cubical regions. The basic techniques is for the query poser to subdivide a geographical region in to a collection of small regions where each consists of half-open cuboids with size $1/2^d$, thereby having a grid of known depth approximate a realistic geometric region. Hence, the query-poser now non-deterministically selects a small region with a peer and sends a request to a small number of regions. The recipient peers in those regions now uses the previous SHA-1 based hashing schema to determine if they are to answer back.

An improvement using finite precision arithmetic (i.e., an **Algorithm 3**) is also described in the paper that selects ONE peer to answer a query, but still satisfy conditions (1) through (4) stated in the requirements.

The main advantage of using this algorithm for MDA is its sensitivity for the geometry. This is so because threat missiles are observed or computed to pass through different regions and sub-regions of the world that has geometric shapes. One of the assumptions used in the paper, that peers exchange messages among themselves when they are not attempting to answer a query is also valid in the MDA domain, as the *Weapons Net* can be used for this purpose. For example, *readiness* and *availability* information can be securely broadcasted in this way.

But we there are some other issues that must be addressed explicitly. That is instead of selecting a leader, can we extend the algorithm to get a leader and a second in command? The objective being, able to recursively run the leader election algorithm so that if the first missile misses the target, or the target disintegrates into two fragments, the second will be ready to fire.

Reviewed Article: Elastic Vector Time [13]

Summary

As the abstract states, this paper 'discusses a means to ensure actions of distributed soft real-time applications are executed in the correct order even in the face of failures.' The authors propose a new time base called Elastic Time Vector, and a synchronization mechanism that is based on two core primitives, *wait()* and *wait-deliver()*.

The elastic time vector is similar to traditional vector clocks in that it is a vector that carries a time stamp for each of the participating processes in the system. It differs from most other vector clocks in that each process time stamp in the vector is synchronized with a low level clock that each process has access to. The idea is to keep the low-level

clocks as close to real-time as possible and let the vector clock slow down for those occasions when processes cannot keep up with processing events. While this approach allows for vector clocks to slow down, it guards against vector clocks slowing down forever through using a synchronization mechanism.

Elastic vector time is defined by five properties: synchronization, scheduling, estimation, unboundedness, and logical time consistency. These properties are defined in such a way that elastic vector clocks can be used for failure detection and communication by time as real-time clocks are similarly used for these purposes in synchronous systems. The performance of each implementation of the elastic time vector can be determined by a Cost Function (CF). The goal of each implementation is to keep the elastic vector time and the low-level clock time as close as possible. The paper states the cost function formally as CF: = $\Sigma_{A \epsilon H}$ (e(A) –ST(A)). In short, the cost function should be minimized, and is represented as the sum of the individual differences between when an action in the set of all actions was executed in real-time and the individual action's scheduled start time. It is important to note that the cost function defines a partial order on the implementation of elastic vector time and is a metric for distinguishing between better and worse implementations. Synchronization is accomplished in elastic vector time by introducing *wait()* and *wait-deliver()* primitives. These primitives, along with two additional vectors *threshold* and *send-time* as well as some of the properties listed earlier, work concurrently to start actions as close to their scheduled start times as possible. They also wait for some small amount of time before giving up on the action and deeming the process crashed. Using these primitives, designers are enabled to lower processing overhead through having the power to choose which actions are necessarily externally consistent, rather than making *all* actions externally consistent.

Discussion

We believe that elastic vector time is not a particularly good candidate for use in the ABM system for the following reason: elastic vector time is specifically designed for soft real-time distributed applications. As the ABM is being designed as a hard real-time system, elastic vector time is simply not usable.

Moreover, although the paper states that the maximum external deviation and drift of the low-level clocks are theoretically not needed in the definition of elastic vector time, in reality they clearly impact the utility of the system (i.e., the closer the synchronization of the low-level clocks, the higher the utility of the system).

Reviewed Article: Eventually Consistent Failure Detectors [21]

Summary

This article introduces a new class of unreliable failure detectors. On top of failure detection, the eventually consistent class of failure detectors implicitly adds an eventual leader election capability. This eventual consistency failure detectors work is based on

seminal work done by Chandra and Toueg [7], which has been used to solve fundamental problems, such as consensus, in asynchronous, distributed systems. In [7], failure detectors are characterized by two properties: completeness and accuracy. As stated in the paper, "completeness characterizes the failure detector capability of suspecting every incorrect process (processes that actually crash) and accuracy characterizes the failure detector capability of not suspecting correct processes." In addition, [7] identifies two types of completeness and four types of accuracy, which result in eight classes of failure detectors when combined. This paper chooses to focus on the four following completeness and accuracy properties: Strong Completeness, Weak Completeness, Eventual Strong Accuracy, and Eventual Weak Accuracy. By combining the above properties in a pair-wise manner, four classes of failure detectors [21], Figure 1) emerges.

In this paper the authors define a new accuracy property and combine it with the strong completeness property to create a new kind of unreliable failure detector termed 'Eventually Consistent.' The interesting property that this class of failure detectors shares is "in each run, all the correct processes eventually converge to the same non-suspected correct process (by means of a deterministic *leader()* function applied to the set of non-suspected processes returned by the failure detector)." This class of failure detectors implicitly provides something close to a leader election mechanism; however, it has shortcomings in that it does not provide knowledge of which node is the leader and it allows for multiple leaders. With this new class of failure detectors, the authors consider the consensus problem and show that the properties of eventually consistent failure detectors allow every correct process to eventually agree on a coordinator that can be used to reach consensus. The remainder of the paper studies the relationship with the other classes of failure detectors. They show, formally, that the eventually perfect class of failure detectors is a subclass of the newly found eventually consistent class, and that the eventually consistent class is a subclass of the eventually strong class of failure detectors. Finally, they show that the eventually consistent class and the eventually strong class are equivalent classes. This is important because the paper goes on to describe implementations of the different classes in order to show that the eventually consistent failure class is as efficient as the eventually strong model. Based on this, the authors presented an efficient algorithm for solving the consensus problem using an eventually consistent failure detector. The authors show how the eventually consistent class allows the algorithm to use a more selective approach to choose a single leader/coordinator and stably reach consensus in a single round versus an eventually strong failure detector, which uses a rotating leader/coordinator paradigm and may require Ω (n) rounds for convergence.

Discussion

This new class of failure detector could potentially be used in the BMDS. This class of failure detectors has the added benefit of an implicit leader election capability. We recommend that this class of failure detector be a candidate among others, at which point we can compare performance though competition in a "fly-off." Of interest would be the comparison of distinct failure detectors and leader election algorithms versus the

eventually consistent algorithm, which can be used for failure detection and leader election.

Reviewed Article: Eventual Leader Election with Weak Assumptions on Initial Knowledge, Communication Reliability, and Synchrony [12]

Summary

From the paper,

> This paper considers the eventual leader election problem in asynchronous message-passing systems where an arbitrary number t of processes can crash (t < n, where n is the total number of processes). It considers weak assumptions both on the initial knowledge of the processes and on the network behavior. More precisely, initially, a process knows only its identity and the fact that the process identities are different and totally ordered (it knows neither n nor t). Two eventual leader election protocols are presented.
>
> The first protocol assumes that a process also knows the lower bound alpha on the number of processes that do not crash. This protocol requires the following behavioral properties from the underlying network: the graph made up of the correct processes and fair lousy links is strongly connected, and there is a correct process connected to t − f other correct processes (where f is the actual number of crashes in the considered run) through eventually timely paths (paths made up of correct processes and eventually timely links). This protocol is not communication-efficient in the sense that each correct process has to send messages forever.
>
> The second protocol is communication-efficient: after some time, only the final common leader has to send messages forever. This protocol does not require the processes to know alpha, but requires stronger properties from the underlying network: each pair of correct processes has to be connected by fair lousy links (one in each direction), and there is a correct process whose output links to the rest of correct processes have to be eventually timely. This protocol enjoys also the property that each message is made up of several fields, each of which taking values from a finite domain.

Discussion

This paper may be overkill for the C2BMC domain. The main assumptions: asynchronous message passing system, and weak assumptions both on the initial knowledge of the processes and on the network behavior are not valid for the C2BMC domain: the C2BMC protocols will have a strict timeout constraints.

Reviewed Article: A New Leader Election Implementation [27]

Summary

From the paper,

> This article introduces a new implementation of a leader election algorithm used in the generic leader behavior known as gen leader.erl. The first open source release of the generic leader contained a few errors. The new implementation is based on a different algorithm, which has been adopted to fulfill the existing requirements. The testing techniques used to identify the errors in the first implementation have also been used to check the implementation proposed here. Additional new testing techniques have been applied to increase confidence in the implementation of this very tricky algorithm. The new implementation passed all tests successfully. This paper describes the algorithm and discusses the testing techniques used during the implementation.

Discussion

This paper provides an excellent introduction into the leader selection problem.

> Many distributed applications are easy to implement if one has one dedicated process to administer certain tasks. For example, one process could poll all attached hardware devices to determine the configuration of a distributed system, whereafter the other nodes may then consult this process for the configuration information. More generally, it is often useful to have a server process that is in charge of keeping a consistent view of an aspect of the system state. All nodes in the distributed system consult that server process if they want information about the system state or if they want to update the system state.
>
> A dedicated server provides an easy way to introduce consensus, synchronization and resource allocation in a distributed system. The disadvantage with this solution is, though, that one introduces a single point of failure in the system. In a fault-tolerant setting, at least one stand-by node needs to be introduced. Taking that thought one step further, several stand-by nodes may be introduced, since that provides an even better protection against faults.
>
> With either one or more stand-by nodes, each stand-by node has the problem of detecting when to become the active node. In fact, the primary node (the one that is assumed to run the dedicated server if nothing goes wrong) also has the problem to determine whether it can actually take that role. This is caused by the fact that when this primary node starts, one of

> the stand-by nodes may already have decided that the primary node is dead and that it should run the server instead.
>
> This problem of having several nodes competing to perform one central task is well-known and described in literature as *the leader election problem*. A solution to this problem is an algorithm that when its execution terminates, guarantees that a single node is designated as a leader and every node knows whether it is a leader or not. The leader is then assigned the role of the above described dedicated server.

There have been an enormous number of publications to this effect. There are over 10, 000 articles on the leader election problem and it is not easy to find a solution among them that fits the given context well.

What is important from the point of view of C2BMC is the fact that each of the suggested algorithms was designed for a completely different situation. Hence, the problem of leader election for the C2BMC should start from a careful analysis of the domain specific features. The paper itself is dedicated to a very special kind of environment called Erlang; it remains to be seen how this environment is related to the C2BMC. The paper also provides a strong caution about applying the proposed leader election algorithms to environments which are different from Erlang.

Reviewed Article: Timeliness, Failure-Detectors, and Consensus Performance [18]

Summary

This paper addresses the reaching of a consensus in distributed computing, which

> …is a widely-studied fundamental problem in theory and practice of distributed systems. Roughly speaking, it allows processes to agree on a common output. The focus here is on the performance of consensus algorithms in different timing models. Although the synchronous model provides a convenient programming framework, it is often too restrictive, as it requires implementations to use very conservative timeouts to ensure that messages are *never* late. For example, in some practical settings, there is a difference of two orders of magnitude between average and maximum message latencies. Therefore, a system design that does not rely on strict synchrony is often advocated; algorithms that tolerate arbitrary periods of asynchrony are called *indulgent*.
>
> As it is well-known that consensus is not solvable in asynchronous systems], the feasibility of indulgent consensus is contingent on additional assumptions. More specifically, such a system may be asynchronous for an unbounded period of time, but eventually reaches a *Global*

Stabilization Time (GST), following which certain properties hold. These properties can be expressed in terms of eventual timeliness of communication links, or using the abstraction of *oracle failure detectors*. Protocols in such models usually progress in asynchronous *rounds*, where, in each round, a process sends messages (often to all processes), then receives messages while waiting for some condition expressed as a timeout or as the oracle's output, and finally performs local processing.

Recent work has focused on weakening post-GST synchrony assumptions, e.g., by only requiring one process to have timely communication with other processes after GST. Clearly, weakening timeliness requirements is desirable, as this makes it easier to meet them. For example, given a good choice of a leader process, it is possible to choose a fairly small timeout, so that the leader is almost always able to communicate with all processes before the timeout expires, whereas having each process usually succeed to communicate with *every* other process requires a much larger timeout. In general, the weaker the eventual timeliness properties assumed by an algorithm are, the shorter the timeouts its implementation needs to use, and the faster its communication rounds can be.

Unfortunately, faster communication rounds do not necessarily imply faster consensus decision; the latter also depends on the number of rounds a protocol employs. A stronger model, although more costly to implement, may allow for faster decision after GST. Moreover, although formally modeled as holding from GST to eternity, in practice, properties need only hold "enough time" for the algorithm to solve the problem (e.g., consensus). But how much time is "enough" depends on how quickly consensus can be solved based on these assumptions. Satisfying a weak property for a long time may be more difficult than satisfying a stronger property for a short time. Therefore, before choosing timeliness or failure detector assumptions to base a system upon, one must understand the implication these assumptions have on the running time of consensus.

This question got little attention in the literature, perhaps due to the lack of a uniform framework for comparing the performance of asynchronous algorithms that use very different assumptions. Thus, the first contribution of this work is in introducing a general framework for answering such questions. Section 2.2 presents GIRAF, a new abstraction of round-based algorithms, which separates an algorithm's computation (in each round) from its round waiting condition. The former is controlled by the algorithm, whereas the latter is determined by an environment that satisfies the specified timeliness or failure detector properties.

The major results indicate that the timely communication with the majority of processes within the system is decisive for optimal performance.

Discussion

As the paper implies it is dedicated to the study of implications of different assumptions on the performance of consensus algorithms for asynchronous distributed systems. Since an ABM system most likely will have strict timeout constraints, and therefore will be a synchronous type of a system, this paper may be of a marginal interest for C2BMC.

Reviewed Article: Distributed Council Election [24]

Summary

The authors develop several algorithms to tackle the problem of electing a small number of representatives (council) out of a group of anonymous candidates. The size of this council is specified in a predetermined range $R = [L, \ldots, U]$. In the special case, these algorithms can be used for election of a single leader by specifying R = [1,…,1]. The authors identify several reasons for why a council could prove useful instead of a single leader: redundancy, as well as better statistical representation if there is a need to poll multiple hosts for some specific type of information.

The following network model is assumed in the presented algorithms. In the election process, all hosts communicate through a central trusted entity (C). Hosts never communicate directly with each other during the election. The election takes place in synchronous rounds. In the first part of the round, C broadcasts a message to all of the hosts. Upon receiving this broadcast message, hosts may or may not send a reply message to volunteer themselves to the council. This decision to volunteer is based on a defined probability. Factors used to evaluate the given algorithms are expected number of rounds for election (T) and expected number of unicast messages needed (N). It is important to note that messages involved in the initial broadcast from C to all hosts are not counted in this message analysis.

The authors first present a naïve algorithm that serves to illustrate the basic probabilistic technique used in all of their algorithms. Using this algorithm (with optimal parameters) in a network with 10,000 hosts and a desired council size in the range of 4-8 members, the expected number of rounds to convergence is 1.43 and the expected number of messages needed is 8.32. This algorithm doesn't perform as well in the case of leader election, often taking more than three rounds to converge. The authors then present the Skip-Reset algorithm. In the case of undershoot (less than L hosts volunteered), this algorithm selects out of all hosts in the next round (like the naïve algorithm). In the case of overshoot (more than U hosts volunteered), hosts that did not volunteer in the current round are dropped from the next round. Figure 10 shows the performance of this algorithm (dotted line) against the number of hosts in the network. The expected number of rounds and messages needed for convergence stay almost constant at ~1.35 and ~8, respectively, even when the number of hosts increases from 100 to 10,000. In the Choice algorithm, hosts draw two coins with a certain probability p in each election round. Hosts

send a message to *C* with both drawings, at which point *C* selects the better of the two rounds. This approach increases the probability of success in a given election round. Although this algorithm requires fewer rounds for convergence, it also requires more messaging, as shown in Figure 11.

In the last part of the paper, the authors discuss robustness to an inaccurate estimation of the number of hosts in the network (*n)*, which affects the probability that hosts use to decide whether or not to volunteer to the council. In the case of underestimation, too many hosts are likely to volunteer in a given election round, resulting in more messages. In the case of overestimation, not enough hosts will volunteer, resulting in more rounds for convergence. However, Figure 15 shows that the presented algorithms are quite robust to these estimation errors. The authors also examine the algorithms' robustness to message loss. Loss of some of the broadcast messages from *C* to the hosts has a minor effect on the algorithms' performance because it will result in an overestimation of *n*. Loss of volunteer messages from hosts to *C* has a minor effect of reducing one of the predefined parameters in the hosts' probability to volunteer equation. This parameter could potentially be adjusted to help compensate for this message loss.

Discussion

There are several positive aspects to using this approach to elect a council in the scope of the C2BMC. The scalability of the presented algorithms ensures that the number of rounds and messages needed for convergence don't increase considerably even when the number of hosts in the network increases by several orders of magnitude. Although this algorithm depends on knowing the number of hosts in the network, it is robust to incorrect estimates. In addition, it is shown that the performance of the algorithms is not greatly affected by some amount of message loss in the network. Message loss is not as carefully evaluated in some of the other papers discussing algorithms for leader election. A unique aspect of this paper is the concept of using a council, rather than a single leader. There may be aspects of the C2BMC that would require the selection of multiple hosts. However, there is no analysis in the paper discussing the efficiency of using the presented algorithms versus executing a leader election algorithm multiple times (could be done in parallel) on different segments of the network in order to elect a council. In fact, one drawback of this paper's approach is due to the probabilistic nature of the algorithms, there is no guarantee that all of the hosts selected to the council are not from the same segment of the network.

In the context of the C2BMC, one significant weakness of this approach is the dependence on a central, trusted entity *C*. We would need to deal with the issue of how *C* itself is selected. One possible solution is to use another leader election algorithm to select *C* and then use the presented algorithms to select a council. However, even if this were done, the availability of *C* would have to be guaranteed during a particular election round. Also, in the C2BMC setting, the initial broadcast messaging by *C* at the beginning of an election round may be too expensive. This expense can be somewhat reduced by maintaining additional state at *C*. Another issue is that the presented algorithms assume that all hosts are equal. Every host volunteers to the council with the same probability. However, in our application, some form of weighting must be used to adjust the

probability of a host volunteering to the council based on the importance of that host to the task at hand. These weightings would have to be carefully normalized across the host population so that the algorithm still had a good chance of electing a correct number of nodes that fell within the predefined range of the council size.

Reviewed Article: An Executable Model for a Family of Election Algorithms [26]

Summary

The key concept presented in this paper is the development of a generative model for a family of similar algorithms can lead to a reusable solution that lowers design costs. In this generative model, algorithm family members are not exhaustively modeled. Instead the generator is configured at its variation points, or 'features,' and in turn outputs the corresponding algorithm family member. A descriptive analogy made in the paper is that such a generator can be thought of as a generator for a car. Whereas the inputs to the algorithm generator are the feature values of the algorithm, the inputs to the car generator are the car's options, such as its color, number of doors, and type of engine. Whereas the output of the algorithm generator is a particular algorithm, the output of the car generator is the requested vehicle.

The authors indicate that they are applying the ideas of system family engineering (SFE) to algorithm development. The major concept of SFE is to 'address variability across a domain in order to maximize reuse.' In the process of creating a generator to address this variability, the authors believe that the algorithm developer will better understand the domain of these algorithms' characteristics. In turn, this domain viewpoint will allow for a better understanding of the differences between existing algorithms, as well as lead to new algorithms that combine the features of existing algorithms.

As an example of their technique, the authors apply this concept to leader election algorithms in a ring network topology. They first identify a feature model for ring leader election algorithms. Features are placed in the following categories: topology, protocol, and node behavior. Topology features relate to the underlying assumptions on the ring network, such as directionality (unidirectional or bidirectional). Protocol features identify how various phases of the election process, such as message passing and election termination should be executed. Node behavior features describe how nodes know that the election is over and which node has been elected as the leader. The authors then identify the need for feature combination rules, which specify certain combinations of features as invalid. Lastly, the authors implement an algorithm generator for ring election algorithms as a state machine, as shown in Figures 1-3. The authors' analysis primarily shows that the state machines formalize an executable method for realizing combinations of the various features. Rational Rose Real Time, made by IBM, was used to implement the state machines.

Discussion

The possible value of this work to the C2BMC could lie in using an algorithm generator to switch between algorithms based on current network conditions. For example, particular algorithms may perform better in the case of a network with lossy links, long delays, or a small number of nodes. Aspects of the network, such as level of packet loss, delay length, and population size, could be specified as feature inputs to the algorithm generator. Based upon manual or automatic specification of these feature points, an algorithm generator could output a leader election algorithm that is best suited to the current network conditions. The performance of such a generator could potentially provide an adaptive approach to the leader election problem that would be interesting to test in simulation. Another possible use of this method is to use an algorithm generator during the development of leader election algorithms for certain tasks in the C2BMC project. This generator would allow us to test various combinations of features present in existing algorithm, which may lead to a hybrid algorithm that better suits our requirements.

Reviewed Article: Time-Free and Timer-Based Assumptions Can Be Combined to Obtain Eventual Leadership [22]

Summary

This paper investigates protocols for implementing the Eventual Leader Oracle for networks with the following assumptions:

(1) At most f (out of N) processors can crash;
(2) Reliable channels - Channels do not create, alter or lose messages;
(3) There is a correct processor p_l (i.e. a processor that is alive) and a set Q of f processors p_x such that $p_l \notin Q$ and $\forall p_x \in Q$, the channel $p_l \rightarrow p_x$ is eventually timely or eventually winning.

The paper takes advantage of the combined time-free and time-based assumptions and presents an eventual leader protocol as follows:

> Each processor p_i to manage an array $count_i[1 : n]$ such that $count_i[j]$ will remain bounded if p_j is correct and p_i trusts it (not to have crashed). Then, given such an array, p_i considers as the current leader the processor p_l such that $count_i[l]$ has the smallest value (when two counters have the smallest value, processor IDs are used to break ties).
>
> To get a $count_i$ array with consistent values, each processor p_i uses two distinct sets of data structures, one addressing the "timely" part of the assumption (3), the other one addressing its "winning response" part.
>
> On the "timely" side, the Boolean array $timely_i[1 : n]$ is the relevant data structure. It is managed as follows: First, each processor p_j sends periodically ALIVE() messages to inform the others that it has not crashed. Accordingly, each processor p_i

manages a timeout value ($timeout_i[j]$) and a timer ($timer_i[j]$) with the hope that it will receive the next ALIVE() message from p_j before that timeout value has elapsed. If a timeout occurs, p_i sets $timely_i[j]$ to false. When it receives an ALIVE() message from p_j, p_i resets the timer and increases the timeout value if $timely_i[j]$ == false. p_i always sets $timely_i[j]$ to true after p_i has received a message from p_j.

So, the idea of the mechanism is to obtain the following property: The processors p_j such that eventually $timeout_i[j]$ remains permanently equal to true determine a set of timely channels $p_j \rightarrow p_i$.

On the "query-response" side, the relevant data structure is the Boolean array $winning_i[1 : n]$. Repeatedly, each processor p_i sends a QUERY() message to all the other processors and waits for the first $(N - f)$ RESPONSE(). The processors that sent these winning responses are defined as the "winning processors" for that query. So, the query-response mechanism is used to get the following property: If a processor p_j is such that, eventually, $winning_i[j]$ remains permanently equal to true, then p_j will no longer be suspected by p_i.

A key element of the protocol is the way the Boolean arrays $timely_i$ and $winning_i$ are combined and used by p_i to update $count_i[1 : n]$. Their combination is done by the condition that p_i trusts the processors p_j such that $timely_i[j] \vee winning_i[j]$. The aim of the set $trusted_i$ is to include the processors that eventually have timely channels towards p_i or whose responses to p_i's queries are eventually always winning.

When a processor p_j sends a response to p_i, it uses the current value of its set $trusted_j$ to inform p_i that it (p_j) does not currently suspect these processors. When it has received the $(N - f)$ winning responses it was waiting for, p_i trusts all the processors trusted by the winning processors and suspects the other processors by increasing their counter accordingly.

Finally, in order for all the correct processors to have the same bounded entries in their $count_i$ arrays (and, consequently, be able to elect the same leader), each QUERY() message piggybacks the $count_j$ array of its sender p_j, thereby allowing the receiver p_i to update its array $count_i$.

Discussion

Any coordinated missile defense requires the establishment and maintenance of a chain of command. Hence, the available of a Leader Oracle is essential to increase the robustness and safety of the C2BMC.

While the concept of eventual leadership presented by Mostefaoui et al is interesting as a fundamental property the Leader Oracle, we need stronger assumptions to obtain tighter bounds on the detection of processor crash and the re-establishment of new leaders. In other words, we need to impose timing constraints on the QoS of the C2BMC network in order to develop time-bounded leader election protocols.

We can use the mechanisms for time-based leadership election, combined with time-bounded channels, to develop time-bounded leadership election protocols.

Reviewed Article: Fast Leader-Election Protocol with Bounded Cheaters' Edge [1]

Summary

This paper focuses on a class of randomized leader election protocols among n distributed processors over a reliable network, where each processor will randomly nominate one of the n processors to be the leader and the processor that receives the most nomination wins the election. If the n processors are truly random (i.e. "honest"), then each processor will have $1/n$ chance to be elected. However, if some the processors are faulty (i.e., "dishonest"), the faulty processors (a.k.a., cheaters) can collaborate among themselves to influence the outcome of the election.

This paper takes an information-theory approach and studies the class of protocols that, given a bounded number of cheaters, can guarantee that the elected leader will be one of the honest processors with at least some fixed nonzero probability. The protocol is assumed to execute in an asynchronous setting, but proceeds in synchronized rounds. The efficiency of a protocol is measured by the number of rounds it takes to produce a leader.

Historically, the effectiveness of such protocol is measured by resilience, which informally translates to "the largest number of cheaters allowed for the protocol to still live up to its guarantee." However, resilience is not useful to determine whether a small number of faulty processors can exercise disproportionate influence on the outcome of the election, or not. Hence, the authors proposed a new measure for the effectiveness of a protocol P, called "cheaters' edge." This measure is denoted by $\text{edge}_P(n, t)$, which describes by what factor malicious processors may increase the probability that any one of them is elected by deviating from the protocol in the midst of t cheaters. For example, if the cheaters' edge is zero, malicious processors cannot change their probability of election; if it is 1, the probability doubles; if it is 2, the probability triples; and so on.

The paper then presents a series of protocols and proofs to show that:

(1) For all $n \geq 4$, $t = 1$, there exists a protocol P that elects a leader with only one round, such that $\text{edge}_P(n, 1) = 0$;
(2) For any n, any $t \geq 2$, and any algorithm P, $\text{edge}_P(n, t) > 0$;
(3) For any n, $t \leq O(n/\log n)$, there is an algorithm to construct a protocol P that elects a leader with 5 rounds, such that $\text{edge}_P(n, t) < K$ for some constant K. Moreover, the algorithm takes polynomial(n) time to construct P;

For any n, $t \leq n$, there is an algorithm to construct a protocol P that elects a leader with polylog(n) rounds, such that $\text{edge}_P(n, t) < K$ for some constant K. Moreover, the algorithm takes polynomial(n) time to construct P.

Discussion

The information-theoretical approach for analyzing the effect of cheaters on randomized protocols does not apply to the framework of the C2BMC since it is highly unlikely to use randomized leader election protocols in missile defense.

Reviewed Article: Efficient Leader Election in Complete Networks [28]

Summary

A leader election algorithm that does not require a *sense of direction*, and chooses a leader in time $O(n)$ by exchanging $O(n)$ messages where n is the number of nodes for complete networks is presented.

Details

The network has n nodes arranged in a directional ring topology. That is every node has a predecessor that sends it messages and a successor that its sends messages to. The messages are in three kinds:

ALG(i): where $i \leq n$ informs the successor node that i is claiming to be a leader
AVS(i): where $i \leq n$ informs the successor that asks for information about the predecessor of i
AVSRESP: that informs that the receiving node can delare itself to be a leader.

The algorithm works in three stages.

1. In the first stage every node is considered passive – that is ready to initiate electing a leader. If a node j receives a ALG(i) message from a node $j < i$, then it passes the information to node i+1.

2. During the second stage, if a node that thinks it is in the running for the leadership receives a message ALG(j) from a node $j > i$, then it declares itself a dummy and passes on the message to its successor.

3. When a node running for leadership get a message AVS from a node $j > i$, then it turns itself a dummy and sends back a message AVSRESP(j)

When a node i receives a message AVSRESP(i), then it declares itself the leader.

The paper show that a every initiation completes after time $O(n)$, exchanging at most $O(n)$ messages and elects a *unique* leader.

Discussion

Although this is deceptively simple, yet an elegant and fast algorithm, it may not satisfy all the hidden requirements of the MDA project. Also the fault and potential misuse models of MDA may not match the assumptions of this paper.

Reviewed Article: Memory space requirements for self-stabilizing leader election protocols [3]

Summary

This paper studies the memory requirements (i.e., the minimum number of states needed in each processor) for self-stabilizing leader election (SSLE) protocols on a unidirectional ring under the control of a centralized daemon (i.e. a centralized scheduler). It considers the following three classes of protocols:

(1) Deterministic protocols on anonymous rings;
(2) Deterministic protocols on id-based rings;
(3) Randomized protocols on anonymous rings;

The authors established the following lower bounds on memory requirements via proof-by-contradictions:

(1) The number of states per processor required by deterministic SSLE protocol under a centralized daemon on unidirectional, prime size and anonymous rings is greater than or equal to the ring size. Hence, the memory requirements grow with the size of the ring. It should be noted that Burns and Pachl have proved that there is no SSLE protocol for unidirectional, non-prime size, anonymous rings under a centralized daemon.

(2) On unidirectional, id-based rings (without a bound on the identifier values), there is no deterministic SSLE protocol under a centralized daemon that uses constant memory space. On id-based rings (whose identifier values are bounded by $N + K$ for the ring size N and some constant K), there is a deterministic SSLE protocol under a centralized daemon using constant memory space only if the ring is bidirectional and prime size. Also, if processors do not change their state once the system is stabilized, then there exists a deterministic SSLE protocol using constant memory space on unidirectional, id-based rings with bounded identifier values.

The number of states per processor required by a randomized SSLE protocol under a centralized daemon on unidirectional, anonymous rings is greater than $(M_N - 2)/2$, where M_N is the smallest integer that does not divide into the ring size N.

Discussion

The memory requirements on the unidirectional ring Leader Election protocols presented by Beauguier et al do not apply to the case of C2BMC because (1) the battle managers are connected by a unidirectional ring and (2) we can safely assumed that the battle managers all have unique IDs that takes log N bits each (where N is the maximum number of battle managers in the system).

THIS PAGE IS INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library, Code 52
   Naval Postgraduate School
   Monterey, CA 93943-5100

3. Research Office, Code 09
   Naval Postgraduate School
   Monterey, CA 93943-5000

4. Dr. Butch Caffall
   Missile Defense Agency
   Washington, DC

5. LTC Jason Stine
   Missile Defense Agency
   Washington, DC

6. Dr. Mikhail Auguston
   Naval Postgraduate School
   Monterey, CA

7. LTC Thomas Cook
   Naval Postgraduate School
   Monterey, CA

8. Dr. Allan Jaworski
   Missile Defense National Team (B)
   Arlington, VA

9. Dr. Doron Drusinsky
   Naval Postgraduate School
   Monterey, CA

10. Ms. Linda Grimaldi
    Missile Defense National Team (B)
    Colorado Springs, CO

11. Mr. Bruce Long
    Missile Defense National Team (B)
    Colorado Springs, CO

12. Dr. Bret Michael
Naval Postgraduate School
Monterey, CA

13. Dr. Thomas Otani
Naval Postgraduate School
Monterey, CA

14. Mr. Scott Pringle
Missile Defense National Team
Crystal City, VA

15. Dr. Man-Tak Shing
Naval Postgraduate School
Monterey, CA

16. Mr. Erik Stein
Missile Defense National Team
Crystal City, VA

17. Ms. Deborah Stiltner
Missile Defense National Team
Crystal City, VA

18. Mr. Mark Thornton
Missile Defense National Team
Huntsville, AL

19. Mr. Tim Trapp
Missile Defense National Team
Crystal City, VA

20. Mr. Harsha Tummala
University of California, Berkeley
Berkeley, CA

21. Dr. Geoff Xie
Naval Postgraduate School
Monterey, CA

22. Dr. Duminda Wijesekera
Naval Postgraduate School
Monterey, CA