Markov Models of Leader Elections in a Smart Grid System

Stephen Jackson

May 15, 2014

1 Introduction

FREEDM (Future Renewable Electric Energy Delivery and Management) is a smart grid project focused on the future of the electrical grid. Major proposed features of the FREEDM network include the solid state transformer, distributed local energy storage, and distributed local energy generation [1]. This vein of research emphasizes decentralizing the power grid, making it more reliable by distributing energy production resources. Part of this design requires the system to operate in islanded mode, where portions of the distribution network are partitioned from each other. The effects of these partitions are still not well understood. This is particularly true in a distributed cyber-physical system, in which partitions may occur in both the cyber and physical domains. Related work[6][13] has indicated that cyber faults can cause a physical system to apply unstable settings.

This work presents a distributed leader election algorithm and Markov model of that algorithm. The presented algorithm maintains the Markov property for the observations of the leader despite omission failures This approach to considering how a distributed system interacts during a fault condition allows for the creation of new techniques for managing a fault scenario in cyber-physical systems. This discussion presents an approach that utilizes Markov chain to model a system's

grouping behavior. These chains produce expectations of how long a system can be expected to stay in a particular state as well as how much time it will be able to spend coordinating and doing useful work over a period of time. Using these measures, the behavior of the control system for the physical devices can be adjusted to prevent faults.

PUT IN A LIST OF SECTIONS

2 FREEDM DGI

This study models the group management module of the FREEDM DGI. The DGI is a smart grid operating system that organizes and coordinates power electronics. It also negotiates contracts to deliver power to devices and regions that cannot effectively facilitate their own needs. DGI leverages common distributed algorithms to control the power grid, making it an attractive target for modeling a distributed system.

The DGI software consists of a central component, known as the broker. This broker is responsible for presenting a communication interface. It also furnishes any common functionality the system's algorithms may need. These algorithms, grouped into modules, work in concert to move power from areas of excess supply to excess demand.

DGI utilizes several modules to manage a distributed smart-grid system. Group management, the focus of this work, implements a leader election algorithm to discover which processes are reachable within the cyber domain. Other modules provide additional functionality, such as collecting global snapshots, negotiating the migrations, and giving commands to physical components.

DGI is a real-time system; certain actions (and reactions) involving power system components need to be completed within a pre-specified time-frame to keep the system stable. It uses a round robin scheduler in which each module is given a predetermined window of execution which it may use to perform its duties. When a module's time period expires, the next module in the line is allowed to execute. The start of each round is synchronized between systems. Each DGI process

will execute the same module at the same time.

3 Group Management

The DGI uses the leader election algorithm, "Invitation Election Algorithm," written by Garcia-Molina [8]. Originally published in 1982, this algorithm provides a robust election procedure that allows for transient partitions. Transient partitions are formed when a faulty link between two or more clusters of DGIs causes the groups to divide temporarily. These transient partitions merge when the link becomes more reliable. The election algorithm allows for failures that disconnect two distinct sub-networks. These sub networks are fully connected, but connectivity between the two sub-networks is limited by an unreliable link.

Many election algorithms have been created. Each algorithm is designed to be well-suited to the circumstances it will deployed in. Specialized algorithms exist for wireless sensor networks [15][7], and other special circumstances [16][11]. Work on leader elections has been incorporated into a variety of distributed frameworks: Isis [4], Horus [14], Totem [12], Transis [3], and Spread [2] all have methods for creating groups. Despite this wide array of work, the fundamentals of leader election are similar across all implementations. Processes arrive at a consensus of a single peer that coordinates the group. Processes that fail are detected and removed from the group.

The elected leader is responsible for making work assignments, and identifying and merging with other coordinators when they are found, as well as maintaining an up-to-date list of peers for the members of his group. Group members monitor the group leader by periodically checking if the group leader is still alive by sending a message. If the leader fails to respond, the querying nodes will enter a recovery state and operate alone until they can identify another coordinator. Therefore, a leader and each of the members maintain a set of processes which are currently reachable, a subset of all known processes in the system.

Leader election can also be classified as a failure detector [9]. Failure detectors are algorithms which detect the failure of processes within a system; they maintain a list of processes that they

suspect have crashed. This informal description gives the failure detector strong ties to the leader election process. The group management module maintains a list of suspected processes which can be determined from the set of all processes and the current membership.

The leader and the members have separate roles to play in the failure detection process. Leaders use a periodic search to locate other leaders in order to merge groups. This serves as a ping / response query for detecting failures within the system. The member sends a query to its leader. The member will only suspect the leader, and not the other processes in their group.

4 Experimental Setup

4.1 Broker Architecture

The DGI software used in this designed around a broker architectural specification. Each core functionality of the system was implemented within a module that was provided access to core interfaces. These interfaces provided functionality such as scheduling requests, message passing, and a framework to manipulate physical devices. The Broker provided a common message passing interface that all modules could access. This interface was then used to pass information between modules. For this purpose of this work, messages are sent as single UDP datagrams. If a datagram is lost, it is not resent. DGI expect an increasing sequence number on datagrams, which ensures message ordering.

4.2 Algorithmic Changes

The original Garcia-Molina algorithm has been modified so the observations of the coordinator process have the Markov property. The full algorithm is presented below. The rest of this section describes the changes that were made to the algorithm and shows how the combination of those changes allow the observations of the Coordinator to follow the Markov process. The execution model of this algorithm assumes a real-time system using a round-robin scheduler. All processes

have their clock synchronized to a reasonable tolerance. At a predetermined time and following predetermined intervals the algorithm executes at each process. Using the synchronized clocks, all processes execute either $Check()$ or $Timeout()$ at the same time. Processes can only form groups if their clocks are sufficiently in sync with another process' clock.

$AllNodes \leftarrow \{1, 2, ..., N\}$

$Coordinators \leftarrow \emptyset$

$UpNodes \leftarrow Me$

$State \leftarrow Normal$

$Coordinator \leftarrow Me$

$Responses \leftarrow \emptyset$

$Counter \leftarrow$ A random initial identifier

$GroupID \leftarrow (Me, Counter)$

function CHECK

This function is called at the start of a round by the leader

if $State = Normal$ and $Coordinator \leftarrow Me$ **then**

$Responses \leftarrow \emptyset$

$TempSet \leftarrow \emptyset$

for $j = (AllNodes - \{Me\})$ **do**

$AreYouCoordinator(j)$

$TempSet \leftarrow TempSet \cup j$

end for

Nodes which respond "Yes" to $AreYouCoordinator$ are put into the $Responses$ set.

When an $AreYouThere$ response is "No" and this process is a coordinator, the querying process is put in the $Responses$ set.

Wait for $Timeout(CheckTimeout)$, Nodes that do not respond are removed from UpNodes.

$UpNodes \leftarrow (TempSet - Responses) \cup Me$

if $Responses = \emptyset$ **then return**

end if

$p \leftarrow \max(Responses)$

if then$Me > p$

Wait time proportional to me-p

end ifMERGE(Responses)

end if

The next call to this is after Timeout(CheckTimeout)

end function

function TIMEOUT

This function is called at the start of a round by the group members

if $Coordinator = Me$ **then return**

elseARE YOU THERE(Coordinator,GroupID,Me)

if Response is No **then** RECOVERY

end if

end if

The next call to this is after Timeout(TimeoutTimeout)

end function

function MERGE(Coordinators)

This function invites all coordinators in Coordinators to join a group led by Me

$State \leftarrow Election$

Stop work

$Counter \leftarrow Counter + 1$

$GroupID \leftarrow (Me, Counter)$

$Coordinator \leftarrow Me$

$TempSet \leftarrow UpNodes - Me$

$UpNodes \leftarrow \emptyset$

for $j \in Coordinators$ **do** INVITE(j,Coordinator,GroupID)

end for

for $j \in TempSet$ **do** INVITE(j,Coordinator,GroupID)

end for

Wait for $Timeout(InviteTimeout)$, Nodes that accept the invite are added to UpNodes

$State \leftarrow Reorganization$

for $j \in UpNodes$ **do** READY(j,Coordinator,GroupID,UpNodes)

end for

$Acknowledge \leftarrow UpNodes$

Wait for $Timeout(ReadyTimeout)$, Nodes that do not acknowledge are removed from UpNodes

$UpNodes \leftarrow UpNodes - Acknowledge$

$State \leftarrow Normal$

end function

function RECEIVEREADY(Sender,Leader, Identifier, Peers)

if $State = Reorganization$ and $GroupID = Identifier$ **then**

$UpNodes \leftarrow Peers$

$State \leftarrow Normal$

Respond Ready Acknowledge

end if

end function

function RECEIVEAREYOUCOORDINATOR(Sender)

if $State = Normal$ and $Coordinator = Me$ **then**

Respond Yes

else

Respond No

end if

end function

function RECEIVEAREYOUTHERE(Sender, Identifier)

if $GroupID = Identifier$ and $Coordinator = Me$ and $Sender \in UpNodes$ **then**

Respond Yes

else

Respond No

Add sender to $Responses$ set for $Check()$ if this process is a coordinator.

end if

end function

function RECEIVEINVITATION(Sender,Leader,Identifier)

if $State \neq Normal$ **then return**

end if

if $Sender \neq 0$ **then return**

end if

Stop Work

$Temp \leftarrow Coordinator$

$TempSet \leftarrow UpNodes$

$State \leftarrow Election$

$Coordinator \leftarrow Leader$

$GroupID \leftarrow Identifier$

if $Temp = Me$ **then**

Forward invite to old group members

for do$j \in TempSet$

$Invite(j, Coordinator, GroupID)$

end for

end if

$Accept(Coordinator, GroupID)$

$State \leftarrow Reorganization$

if $Timeout(ReadyTimeout)$ expires before $Ready$ is received **then**

$Recovery()$

end if

end function

function RECEIVEACCEPT(Sender,Leader,Identifier)

if $State \leftarrow Election$ and $GroupID = Identifier$ and $Coordinator = Leader$ **then**

$UpNodes \leftarrow UpNodes \cup Sender$

end if

end function

function RECEIVEREADYACKNOWLEDGE(Sender)

$Sender$ is removed from $Acknowledge$ in $Merge()$

end function

function RECOVERY

$State \leftarrow Election$

Stop Work

$Counter \leftarrow Counter + 1$

$GroupID \leftarrow (Me, Counter)$

$Coordinator \leftarrow Me$

$UpNodes \leftarrow Me$

$State \leftarrow Reorganization$

$State \leftarrow Normal$

end function

In a distributed system information cannot be instantaneously spread throughout the system. A process can only make local observations. In this work, we attempt to model what a process will observe as a result of omission failure. Therefore, it is important that observations that a process makes hold to the Markov property. In the original algorithm, there are several portions that, when projected to the leader's observation, do not meet the Markov property. The following sections state the portions of the algorithm where the observation of the leader process does not yield the probability of next transition.

Leader selection is performed a priori– only process 0 may become a group coordinator. Only process 0 may become the leader of a multiprocess group. This simplification was applied because the configuration of the system with a larger number of processes depended on the configuration of the other processes. Without this simplification, the state of the rest of the system would not have the memoryless property. The state of the processes that are not in the observers group would change each round. As a consequence the state of the rest of the system and the likelihood of forming a specific group size would change each step if other processes could become leader.

DIAGRAM HERE

The changes added a third message to completing an election – a ready acknowledge message. This message is sent by a member after receiving the ready message from the coordinator. This allows the coordinator to be certain of the member's status before the next round. Without the ready acknowledgment, the member may not receive the ready message and the coordinator will observe the member is a part of the group. As a consequence of that uncertainty, the probability that a member remains in a group in the first round after an election has a different probability than each subsequent round. By adding the extra message, the observation of the coordinator of the

state, must be the state of the member of the group. The sequence presented in figure XXX is not possible and as a consequence, the probability a member remains in a group in the first round after an election is a fixed value.

DIAGRAM HERE

Members cannot leave a group without the leader's permission. Members do not suspect the coordinator has failed, only the coordinator may suspect the members. For the purpose of starting an election, an Are You There message and it's negative response are considered equivalent to a Are You Coordinator message and a positive response. On receipt of the negative response, the member will immediately recover and become a leader. This assumption relies on Are You Coordinator and Are You There messages being sent at roughly the same time.

DIAGRAM HERE

This change leads to a live-lock situation in a crash failure, where the group's leader crashes and does not return and as a consequence the remaining members are trapped in a group without a leader. For the purpose of this work, we have disregarded these live lock scenarios. However, the live-lock could be avoided if the member can detect that it has not received an "Are You Coordinator" message in a round. When the process fails to receive the message, the coordinator must have also removed the member from their group since they could not have received a "Are You Coordinator" response message. Integrating this component is an area of future work.

5 Formal Modeling

5.1 Assumptions

All participating peers were assumed to be on the same schedule; all peers began executing the model simultaneously. Synchronization was accomplished using Choi's work [5]. It was also assumed the clocks are synchronized. If the network has faulted, process clocks would not drift noticeably from their last synchronization. As an alternative, a production system would likely use GPS time synchronization to obtain certain power system readings [10].

5.2 Constructing The Markov Chain

6 Conclusion

References

[1] R. Akella, Fanjun Meng, D. Ditch, B. McMillin, and M. Crow. Distributed power balancing for the freedm system. In *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*, pages 7–12, Oct 2010.

[2] Y. Amir, C. Danilov, M. Miskin-Amir, J. Schultz, and J. Stanton. The spread toolkit: Architecture and performance. Technical report, Johns Hopkins University, 2004.

[3] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Transis: a communication subsystem for high availability. In *Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on*, pages 76–84, 1992.

[4] K. Birman and Renesse R. *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press, Los Alamitos, CA 90720-1264, 1994.

[5] Bong Jun Choi, Hao Liang, Xuemin Shen, and Weihua Zhuang. Dcs: Distributed asynchronous clock synchronization in delay tolerant networks. *Parallel and Distributed Systems, IEEE Transactions on*, 23(3):491–504, March 2012.

[6] A. Choudhari, H. Ramaprasad, T. Paul, J.W. Kimball, M. Zawodniok, B. McMillin, and S. Chellappan. Stability of a cyber-physical smart grid system using cooperating invariants. In *Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual*, pages 760–769, July 2013.

[7] Qi Dong and Donggang Liu. Resilient cluster leader election for wireless sensor networks. In *Sensor, Mesh and Ad Hoc Communications and Networks, 2009. SECON '09. 6th Annual IEEE Communications Society Conference on*, pages 1–9, June 2009.

[8] H. Garcia-Molina. Elections in a distributed computing system. *Computers, IEEE Transactions on*, C-31(1):48 –59, January 1982.

[9] C. Gomez-Calzado, M. Larrea, I. Soraluze, A. Lafuente, and R. Cortinas. An evaluation of efficient leader election algorithms for crash-recovery systems. In *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*, pages 180–188, 2013.

[10] A. P S Meliopoulos, G.J. Cokkinides, O. Wasynczuk, E. Coyle, M. Bell, C. Hoffmann, C. Nita-Rotaru, T. Downar, L. Tsoukalas, and R. Gao. Pmu data characterization and application to stability monitoring. In *Power Engineering Society General Meeting, 2006. IEEE*, 2006.

[11] N. Mohammed, H. Otrok, Lingyu Wang, M. Debbabi, and P. Bhattacharya. Mechanism design-based secure leader election model for intrusion detection in manet. *Dependable and Secure Computing, IEEE Transactions on*, 8(1):89–103, Jan 2011.

[12] L.E. Moser, P.M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, and C. A. Lingley-papadopoulos. Totem: A fault-tolerant multicast group communication system. *Communications of the ACM*, 39:54–63, 1996.

[13] T. Paul, J.W. Kimball, M. Zawodniok, T.P. Roth, B. McMillin, and S. Chellappan. Unified invariants for cyber-physical switched system stability. *Smart Grid, IEEE Transactions on*, 5(1):112–120, Jan 2014.

[14] Robbert Van Renesse, Takako M. Hickey, and Kenneth P. Birman. Design and performance of horus: A lightweight group communications system. Technical report, Cornell, 1994.

[15] M.M. Shirmohammadi, K. Faez, and M. Chhardoli. Lele: Leader election with load balancing energy in wireless sensor network. In *Communications and Mobile Computing, 2009. CMC '09. WRI International Conference on*, volume 2, pages 106–110, Jan 2009.

[16] S. Vasudevan, B. DeCleene, N. Immerman, J. Kurose, and D. Towsley. Leader election algorithms for wireless ad hoc networks. In *DARPA Information Survivability Conference and Exposition, 2003. Proceedings*, volume 1, pages 261–272 vol.1, April 2003.