Leader election algorithms: History and novel schemes

Mina Shirali
Member of young researchers club
Islamic Azad University of Qazvin
Mina_shirali@yahoo.com

Abolfazl HaghighatToroghi
Leader of IT department
Islamic Azad University of Qazvin
haghighat@qazviniau.ac.ir

Mehdi Vojdani
Member of young researchers club
Islamic Azad University of Qazvin
Mehdi_vojdani@yahoo.com

Abstract

This Days networks are growing rapidly and managing this networks becomes harder too. A similar case can be seen in the leader election area. leader election is the process of designating a single process as the organizer of some task distributed among several nodes. first we discussed about primary leader election algorithms (bully and ring) and their improvements then considering some assumptions we have proposed two new schemes and discussed about some aspects of them.We proposed Sun(n) and Divided(n) algorithms that can be seen as a tradeoff between bully and ring algorithms. Note that if proposed algorithms didn't work then the original algorithm will be done. That is this algorisms are optionally and just try to reduce latency in the vast networks. In general our paper provide a srvey and a good vision af designing leader election algorithms.

1. Introduction

In distributed computing, leader election is the process of designating a single process as the organizer of some task distributed among several computers (nodes). In many cases we need a coordinator in the network for coordination tasks. When this coordinator crashes, we have to select another process as the substitute. Here leader election algorithms appear. In this paper we discussed about main leader election algorithms and some of their aspects in the sections 1,2 and 3. then their improvements are explained. However our main centralization is on the ring algorithm. Two new schemes are proposed too in the sections 4 and 5. In the section 7 conclusion and comparisions are provided too. Reading this paper gives a good vision for designing a good algorithm depending on your conditions.

2. Bully Algorithm

Suppose one process P[i] detects that the coordinator has crashed and now we require to find new coordinator for coordination operation in system. All of the processes have one process number. Bully algorithm for choosing the best substitute (biggest process number), follows this steps:

a) P[i] sends "ELECTION" message to all processes that have bigger number than P[i].
b) Each process that is active and it's number is bigger than P[i], will send a "OK" message to P[i].
c) If P[i] has received even one "OK" message, it will know that it can not be the coordinator else it will repeat steps a to c.
d) Finally one process that has bigger number than others and hasn't received any "OK" message, announce itself as new coordinator to all of the processes by "COORDINATOR" message.

Suppose that we have n active processes in a distributed system and middle process with process number $\lfloor n/2 \rfloor$ detects that coordinator has crashed. Assume that mn(i) is total number of messages that process p[i] sends ("ELECTION" message) and receives ("OK" message) after sending :

Number of messages =
$mn(1) + mn(2) + \ldots + mn(\lfloor n/2 \rfloor - 1) + mn(\lfloor n/2 \rfloor) + mn(\lfloor n/2 \rfloor + 1) + \ldots + mn(n)$
$= 0+0+\ldots+0+ mn(\lfloor n/2 \rfloor) + mn(\lfloor n/2 \rfloor + 1) + \ldots + mn(n)$
$= (\lceil n/2 \rceil + \lceil n/2 \rceil) + ((\lceil n/2 \rceil - 1) + (\lceil n/2 \rceil - 1)) + \ldots + 2 + 0$
$\cong n + (n-2) + \ldots\ldots + 0 \quad \Rightarrow O(n^2)$

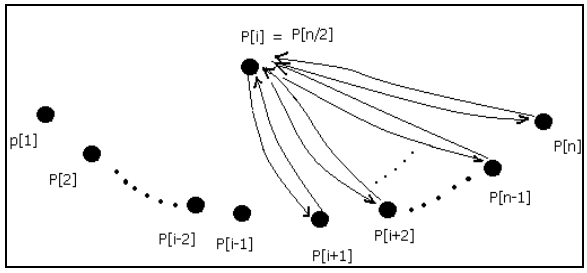

Figure 1. Primary bully algorithm

Steps are:
1-P[i] sends "ELECTION" to greater ones

978-0-7695-3407-7/08 $25.00 © 2008 IEEE
DOI 10.1109/ICCIT.2008.57

2-P[j] sends "OK" to P[i] where j>=i
3-P[j] sends "ELECTION" to greater ones
4-P[j] receives "OK" from greater ones that are not down
5- greatest one broadcasts "COORDINATOR" message
=> we have 5 steps and latency = 1+1+1+1+1=5

2.1. Improvement

In the step b of this algorithm processes can send their numbers to p[i](instead of "OK" message) so in the step c, process P[i] can choose biggest number (considering received numbers) and announce it as coordinator by broadcasting the "COORDINAOR" message. So in the fist step of this algorithm p[i] will send an "ELECTION" message, then it will receive bigger numbers of active processes, then it will select biggest one and announce it as coordinator by sending a "COORDINATOR" message. Considering this steps and previous assumptions the

Number of messages=$\lceil n/2 \rceil + \lceil n/2 \rceil + n \cong 2n$ => O(n)
Number of steps=3
Latency = 1+1+1=3

3. Token Ring Algorithm

In this algorithm, similar to the bully algorithm, each process has one process number. but we have one difference here. In token ring algorithm each process in the ring must save addresses (ring's map).

Suppose that process P[i] detects that coordinator has crashed and now it want to find new coordinator for coordination operation in the system by ring algorithm. This algorithm works as below:

a) Process P[i] sends one token with "ELECTION" title to the next process in the ring. (to P[i+1]).

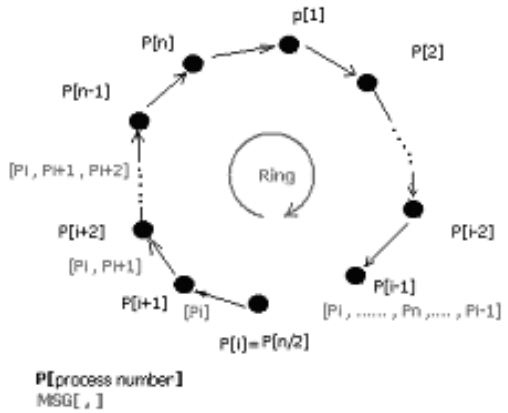

Figure 2. Primary ring algorithm

b) If next process doesn't reply, then process P will suppose that next process is down and will send "ELECTION" message to second next process in the ring (to P[i+2]). P[i] repeats this works until finds next process that is not in down status and has Recieved the "ELECTION" message. Founded process repeats this work too, and this will go on till election message returns back to the process P[i]. each process when receives "ELECTION" message; before forwarding that, puts it's process number in that message.
c) When the "ELECTION" message returns back, then P[i] will choose the biggest process number as coordinator(between process numbers in the "election" message). then it wheels a "COORDINATOR" message in the ring to announce new coordinator.

In this case, we have two round of "message passings" in the ring. First for "CRASH" and "ELECTION" massages and second for "COORDINATOR" message. So we have 2 step and the number of messages and average time are as below:
Number of messages = 2*n =2n => O(n)
Latency = 2*n =2n => O(n)

Note that each process adds it's number to the "ELECTION" message, so when number of processes increase (specially in the end of ring), overheard increases too. For solving this problem we edit this algorithm as each process adds its number to the "ELECTION" message as "biggest" only when it's number is bigger than "biggest" number in the message. In another word we can send only biggest one instead of all of the numbers.

Now assume that more than one process detect that coordinator is crashed and send token in the ring. In this case we have additional overhead and even consistency may be refused. For solving this problem we can change algorithm as; when one of this processes(that has sent an "ELECTION" message), receives message that is sent from other ones it will follow this rules:

a) If the number of this process is bigger than the number of sender process, it will kill the message.
b) If the number of this process is less than sender's number, it will let this message go round.

4. Sun Algorithm Idea

Assume that we just can forward messages clockwise or we don't want to use duplex links. In the sun ring algorithm similar to the ring algorithm we have a ring, but this ring is divided into m multiple subrings. We can divide the ring with n process to the d subrings (d<n). however total number of messages in this algorithm is not less than original ring algorithm but the overal latency is reduced. this algorithm for choosing new coordinator passes the below steps:

a) P[i] which has detected that the coordinator has crashed; wheels the "election" message (token) with its process number as "starter" in the ring.

b) When a process that its numbers is a multiple of m (i mod m =0, denoted by Pm) receives the "election" message, it starts wheeling this message in the ring of the processes that their numbers are a multiple of m (denoted by Pm ring). This message contains first Pms number that has forwarded message as "informer".
c) Each Pm when receives "election" message, starts wheeling an "election" message with its number as "subring-biggest" to the next process in the related subring.
d) Each process that is not a Pm when receives "election" message, if it's process number is bigger than "biggest", adds it's number to this message as (in place of) "biggest" and then forwards it to the next process.
e) When the "election" message rerurns back to the informer (after wheeling in all of the rings), it kills this message and sends a "M-coordinator" message to the next Pm with its Number as the "ring-biggest" too.
f) Each Pm when receives "election" message, kills the "election" message then considering "subring-biggest" in the "election" message and its own number it will choose the biggest one as "subring-biggest". then it waits for "M-coordinator" message.
g) Each Pm that has received both "election" and "M-coordinator" messages, compares "ring-biggest" with "subring-biggest" and chooses biggest one as "ring-biggest". After that it will pass "ring-biggest" to the next Pm with "M-coordinator" message.
h) When "M-coordinator" message, received this message returns back to the informer, it will get "ring-biggest" and kills this message. then starts wheeling a "coordinator" message with "ring-biggest" as coordinator in a sun ring method.(i.e. wheeling in the ring of **Pm**s and Each Pm will wheel this message into its subring too).

In the figure 3 it is shown that how this algorithm works and messages are numbered. For example 1e means sending "election" message in the step 1 and 3Mc means sending "M-coordinator" message in the step 3 and 22C means sending "coordinator" message in the step 22. As we described, we have 3 step here that is wheeling this massages:

a)"election" message(in the Pm ring and subsequently in the subrings)
b)"M-coordinator" message (in the Pm ring)
c)"coordinator" message (in the Pm ring and subsequently in the subrings). so number of messages and latency are as below:

Number of messages = (m+n)+m+(m+n) => O(m+n)

For computing the overall latency we have to compute it in the last subring:

1- receiving "election" message from the Pm ring and forwarding it to the infomer through subring = (m)+(n/m)
2-"M-coordinator" returns back to the informer and last comparison take place (m comparisons takes place) = latency in step one+m
3- informer starts wheeling a "coordinator" message after steps one and two that is (m+n/m)+(m).
4- "coordinator" message returns back to the infomer through subring after step 3= (2m+(n/m)) + (m+n/m)

So **latency = 4(m)+2(n/m)=> O(m+(n/m))**

Now assume that more than one process (each one is denoted by Pcrash) detect that coordinator is crashed. It can lead in overhead, latency or even unconsistency; for solving this problem, each Pm that has received "election" message from more than one informer follows this rules**:**

a. Each informer starts sun ring algorithm only towards the first Pcrash and kills "election" messages after that.
b. Each Pm that has started sun ring algorithm as informer, will kill other informer's messages and sends received information to the biggest informer through Pm ring.
c. In the rule b, only when we have new informations sending operation will take place.

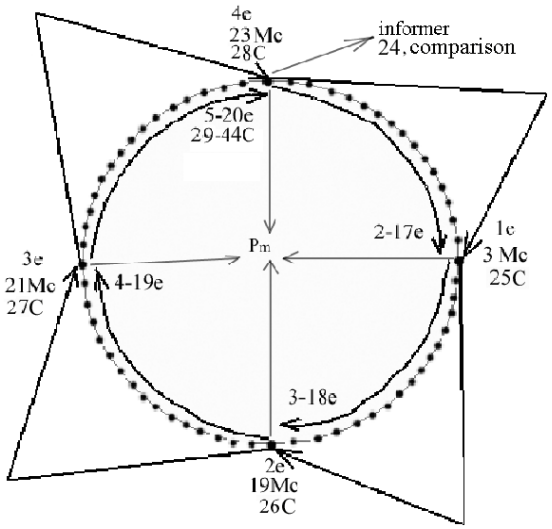

Figure 3. Sun(1) ring algorithm(u=16)

5. Sun(n) ring Algorithm

Now we can extend and use this method again and again till internal ring's length is greater than u (Assume that we use a fix constant(u) in divisions). each time we divide the internal ring (iTH ring), new ring(R_{i+1}) and subrings(sR_i) will be made. Each subring from ring i has u nodes of ring i and with connecting division points (end points of a ring's subrings) we have next internal ring('$i+1$'TH ring) that its length is: $L_{i+1}=(L_i)/u$

For example assume that first division length (L1) is 128 and u is 4 then next divisions will have this lengths (number of connections): L2 =64, L3=16, L4=4. as you see, in this case each L is an exponent of u.

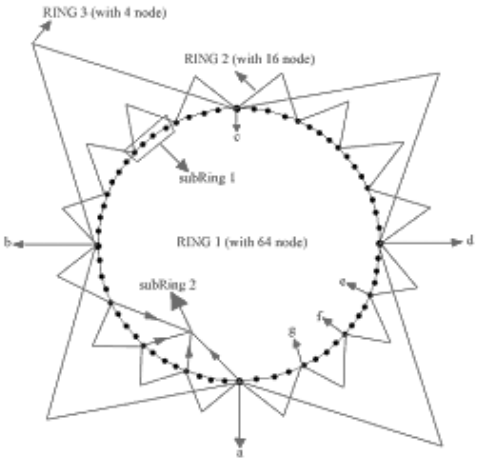

Figure 4. Sun(2) ring algorithm(u=4)

In the following we have extended this method once more(Sun**2**) with a fix constant in divisions. Here the number of nodes in iTH ring is dentoted by Li. Below you can see an example. In the figure 4, first we have a ring with 64 points(L1=64), then we use constant 4 for making division that will reult a ring with 16 nodes (L2=16). Note that in the figure 3 our constant (u) is 16. Again we divide the internal ring (with 16 node) and make a ring with 4 nodes(L3=4). As you see, now it is look like the sun.

Assume that wheel(X,Y) means wheeling message "X" in the "Y" and sRi means subrings of iTH ring and Ri means iTH ring. Steps are :

a) wheel(election, R3)
b) wheel(election, R2)
c) wheel(election, R1)
d) wheel(M-coordinator, R3)
e) wheel(coordinator, R3)
f) wheel(coordinator, R2)
g) wheel(coordinator, R1)

As you see we have 8 steps here and in sun(n) when we have r rings(r=n+1), number of messages and latency can be computed as below (according sun(2)):

Number of messages = (L3 + L2 + L1) + L3 + (L3 + L2 + L1) = 2(L3+L2 +L1)+L3

So number of messages = $2*\sum_{i=1}^{r} L_i + L_r$, Li=L/u^i

Here we have to pass three messages (election, M-coordinator, coordinator). Assume that node "a" is an informer and messages are passing clockwise. Most latency will be seen in the last parts. So for computing the average time we have this steps:

Step 1:
1- "election" message returns back to the Point "a" after wheel(election,R3)+ wheel(election,sR2)+ wheel(election,sR1) = L3 + u + u =>Lr+(r-1)*u

Step 2:
2- "M-coordinator" message returns back to the Point "a" after passing step one and (L3) comparison =2*(L3)+2u. =>2*L3+(r-1)*u

Step 3 (after maximum time between step one and two):
3- Point "a" will wheel "coordinator" message in all of the rings that is latency in step two+L3+2u =3*(L3)+4u =>3*(L1/(u^m))+2*(r-1)*u.

so latency = 3*(L1/(u^m))+2*(r-1)*u

Here we reduced latency. Note that in the bully algorithm all of the processes that have bigger number of P[i], send their information to the P[i], but in the sun ring algorithm only next and previous nodes in the ring or subrings communicate with informer. so we have less number of collisions towards the bully algorithm (when number of processes is large and i is a small number).

6. Divided ring Algorithm

Now we want to relax the "duplex link" assumption. In this algorithm we have a node as informer that can inform some points of the ring and improve ring algorithm's speed. In the figure 5 an informer with 4 point is shown.
This algorithm works as below:

a) Each node that has detected crashing of the coordinator, wheels "election" message in the main ring (like the original ring algorithm)
b) Each node that is a point of informer and has detected the crashing or received the "election" message, send "crash" message to the informer.
c) Informer informs its points (division points) with sending "election" message to all of its points
d) Each point that has received the "election" message and is not down, will send a **reply back** to the informer.
e) If informing received more than one reply, it sends a "**divide**" message to the associated points.
f) Each point that has received the "divide" message(denoted by Di), if it has not sent "**election**" message yet, now it wheel this message in its subRing.
g) Each Di that has received election message, kills this message and **gives this information to the informer**.
h) When informer gets information of all Di's, it will select the best one according received informations

and will send a "**coordinator**" message (with new coordinator number) to all of the Di's.

i) Each Di wheels this "**coordinator**" message in its subring, so all of the ring nodes will know about new coordinator.

As you see number of steps (since informer is informed about crashing) is 7, and after sending "divide" message; latency and messages are: (Assume that number of points is P, number of points that are not down is D and number of ring nodes is L1)

Message =D+(L1)+D+D+(L1) ≅ 3D+2 *(L1)

Time = 1+(L1/D)+1+1+(L1/D) = 3+(2 *(L1)/D)

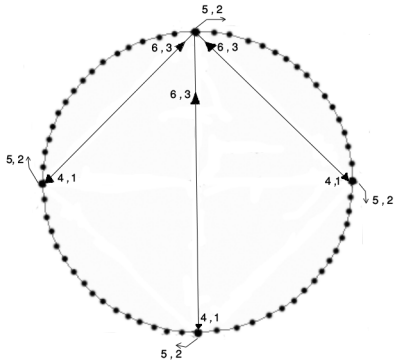

Figure 5. Divided(1) ring algorithm(u=4)

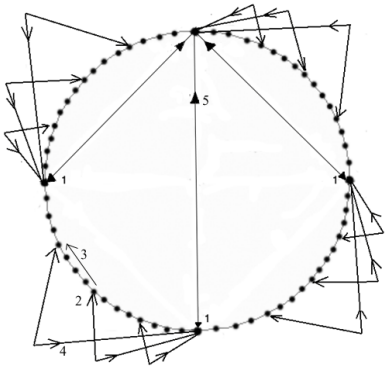

Figure 6. Divided(2) ring algorithm(u=4)

We can use this algorithm again and divide the subrings too. In general two rounds of message passing must be executed. in the figure 6 it is shown that how a message wheels in the partitions according d(2) ring algorithm (e.g. wheeling "coordinator" mesage).

For better performance and less collisions we can make divisions with different lengths or use backoff time strategy that is each node before sending its message have to wait for a random time and then it can send.

Note that with this algorithm we has reduced number of collisions towards bully algorithm. But it will be usefull for a ring with high number of nodes. otherwise collisions and subsequently latency will increase.

The interesting part of this algorithm is that; if informer is down, original ring algorithm will be done and with using informer, we just try to improve ring algorithm's speed.

Assume that we have a constant number in division (u) that specify our partition numbers and original ring's length is L1 so each time we are dividing a ring or a subring into u parts. So with first division we have u subrings (partitions) that each one have (L1/u) length (L2=L1/u). In the second division we have u partition in each part that each one have L2/u length. In another word now we have u^2 part that each one have L1/(u^2) length. Ignoring collisions the division process can continue till partition length is big enough and division improves the performance. So in mTH division :

Lm = L1/(u^m)

Number of partitions = u^m

Number of messages for making u partitions = u-1

Number of messages = two round of maessage passing; first for wheeling "elction" message and the second one for wheeling "coordinator" message

≅ 2 L1+3 (u^m)

Latency=m+Lm+m+m+Lm=3m+2Lm=3m+2*L1 / u^m

We can use a combination of sun and divided ring algorithms too. For example in the divided ring algorithm in the step m and p we can wheel the "election" and "coordinator" messages in the related subrings using sun technology. It is shown in the figure 7.

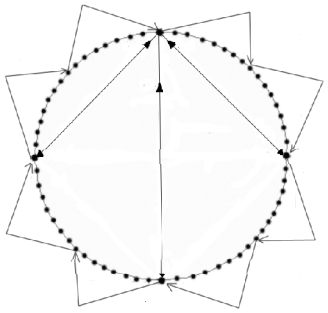

Figure 7. A combination example

7. conclusion

This paper gave you a good vision about leader election algorithms and designing issues. Now depending on your condition, you can design the best architecture. For example now you know that in the small networks, bully algorithms is the best one.

However if you have a ring network, you choose the ring algorithm and if your network is vast and has a lots of nodes, considering number of collisions and depending on the conditions like position of nodes, the distance between them and duplexing of the links you can use one of the above algorithms or a combination of them. Below

you can see a conclusion of this paper and a comparison between leader election algorithms:

Table 1. A comparison between leader election Algorithms

	messages	Latency
Bully	O(n)	3
Ring	O(n)	O(n)
S(n)	$2*\sum_{i=1}^{r} L_i + L_r$, Li=L/u^i	3*(L1/(u^n)) + 2*(r-1)*u
D(n)	2 *L1+3 (u^n)	3n+2*(L1/(u^n))

Table 2. Computed number of Messaes and latency

With 64 nodes	Assumption	messages	Latency
Bully	first P[32] detects that coordinator is crashed	128	3
Ring	---	128	128
S(1)	We have 4 division points(Pm) u=4	176	64
D(1)	We have 4 division points(Pm) u=4	140	36
S(2)	u = 4	172	28
D(2)	u=4	176	14

7. References

[1] S. Tanenbaum; M. V. Steen, "DISTRIBUTED SYSTEMS: Principles and Paradigms", 2e, Prentice Hall, Inc, 2007

[2] EffatParvar, Mehdi; Effatparvar, MohammadReza; Bemana, Akbar; Dehghan, Mehdi," Determining a Central Controlling Processor with Fault Tolerant Method in Distributed System", ITNG apos;07, 2-4 April 2007 Page(s):658 – 663.

[3] Heutelbeck, D.; Hemmje, M," Distributed Leader Election in P2P Systems for Dynamic Sets", MDM 2006. 7th May 2006 Page(s): 29 – 29.

[4] Francis, P.; Saxena, S, "Optimal distributed leader election algorithm for synchronouscomplete network ",TENCON apos;98., 1998 Page(s):86 - 88 vol.1

[5] Sudarshan Vasudevan; Brian DeCleene; Neil Immerman; Jim Kurose; Don Towsley, "Leader Election Algorithms for Wireless Ad Hoc Networks", DARPA Information Survivability Conference and Exposition, April 2003 Page(s): 261 - 272

[6] Neeraj Jaggi; K. Gopinath, "Verification of a Leader Election Algorithm in Timed Asynchronous Systems", FSTTCS 2001, Page(s): 207-218

[7] E.H. Kim; J. K. Kim, "A leader election algorithm in a distributed computing system", FTDCS 1995, Page: 481 .

[8] Zargarnataj, M, "New Election Algorithm based on Assistant in Distributed Systems", AICCSA 2007, May 2007 Page(s):324 – 331.

[9] Sung-Hoon Park, "A Probabilistically Correct Election Protocol in Asynchronous Distributed Systems", Springer Berlin , Heidelberg, 2003.

[10] Sung-Hoon Park; Yoon Kim; Jeoung Sun Hwang, "An efficient algorithm for leader-election in synchronousdistributed systems", TENCON 99,Dec 1999 Page(s):1091 - 1094

[11] Attiya, H., Welch, J.: Distributed Computing, Fundamentals, Simulations, and Advanced Topics. McGraw-Hill Publishing Company, UK 1998.

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