Determining a Central Controlling Processor with Fault Tolerant Method in Distributed System

Mehdi EffatParvar[1], MohammadReza Effatparvar[2], Akbar Bemana[3], Mehdi Dehghan[4]
[1]*Islamic Azad University of Ardebil, Ardebil, Iran*
[2]*Young Researcher Club, Islamic Azad University of Qazvin, Qazvin, Iran*
[3]*Islamic Azad University of Hashtroud, Hashtroud Iran*
[4]*Computer & IT Eng Deportment of Amir Kabir University, Tehran, Iran.*
E-mail: Mehdi_Effatparvar@yahoo.com

Abstract

Central Controlling Processor is applied in many scientific fields such as computer network, centralized mutual exclusion algorithm, centralized control IPC, Berkeley algorithm, etc. Central Controlling Processor in distributed systems is a very important problem, and this problem must be solved by suitable algorithms. The main goal of Central Controlling Processor is synchronizing the process at optimal using of the resources. In this paper we call the Central Controlling Processor as a leader many different algorithms have been presented for leader election. The most important leader election algorithms are the Bully and Ring algorithms. Ring election algorithm is one of the classic method which is used to virtual ring and determine the process with highest number as the coordinator, and one of the most important leader election algorithm is the Bully algorithm. In this paper we will describe novel approaches with fault tolerant method to improve the Bully and Ring algorithms. Our simulation shows that our algorithm is more efficient rather than the Ring algorithm in number of message passing. By doing this, performance and behavior will be improved and message passing will be reduced.

1. Introduction

Leader election in distributed systems is a very important problem that it solves by using suitable election algorithms. In election algorithm we intend to elect a single coordinator for some processes in distributed systems. In distributed systems, processors communicate with each other using shared memory or by exchanging messages with each other. For processors to perform any distributed task effectively the processors require coordination. In a pure distributed system, there is no central controlling processor that arbitrates decisions. Without a central authority or coordinator, any processor has to communicate with all processors in the network to make decision. Often during the decision process, not all processors make the same decision. Communication between processors takes time and further more, making the decision takes time. Coordination among processors becomes difficult when consistency is needed among all processors. Centralized controlling processor(s) can be selected among the group of available processors to reduce the complexity of decision making. Many distributed algorithms require one process to act as coordinator, initiator, or otherwise perform some special role. In general, it does not matter which process takes on this special responsibility, but one of them has to do it.

Leader election is a technique that can be used to break the symmetry of distributed Systems. By determining a central controlling processor (leader) in the distributed systems a processor is elected as the leader among the group of processors in the distributed systems. This processor acts as the centralized controller of this decentralized distributed system.

Some applications of leader election include finding a spanning tree with the elected leader as root[19], breaking a deadlock , reconstructing a lost token in a token ring network, using leader election in Ad Hoc network [17,18].

Leader election algorithms for static networks are popular. These algorithms work by constructing several spanning trees with a prospective leader at the root of the spanning tree and recursively reducing the number of spanning trees to one. However, these algorithms work only if the topology remains static and hence cannot be used in a mobile setting.

The purpose of leader election [1] is to choose a processor that will coordinate activities of the system. In any leader election algorithm, a leader is usually decided based on some criterion such as choosing the processor with the largest identifier as the leader. At the time when the leader is decided, the processors reach the terminated states. The terminated states, in a leader election algorithm, are partitioned into elected states and non-elected states. When a processor enters a non-elected state (or an elected state), the processors always remain in the non-elected state (or an elected state). Any leader election algorithm must be satisfied by the safety and liveness condition for an execution to be admissible. The liveness condition states that every processor will eventually enter an elected state or a non-elected state.

The safety condition for leader election requires that only a single processor can enter the elected state. This processor becomes the leader of the distributed system. Several leader election algorithms have been proposed over the years [2-12]. Some of the grand election algorithms that we can mention to them are Bully algorithm, Ring algorithm, Chang and Roberts' algorithm [13], Peterson's election algorithm [16], Lelann's algorithm [15], Franklin's algorithm [14]. Such leader election algorithms proposed until now require processors to be directly involved in leader election. Information is exchanged between processors by transmitting messages to each other. The processors exchange messages with each other and try to reach an agreement. Once an agreement is reached, a processor will be elected as leader and all other processors will acknowledge the presence of the leader.

In this paper we present new approach for Bully algorithm with fault tolerant , that it decrease the message complexity of Bully, and also a new approach for decreasing the message in Ring algorithm is represented.

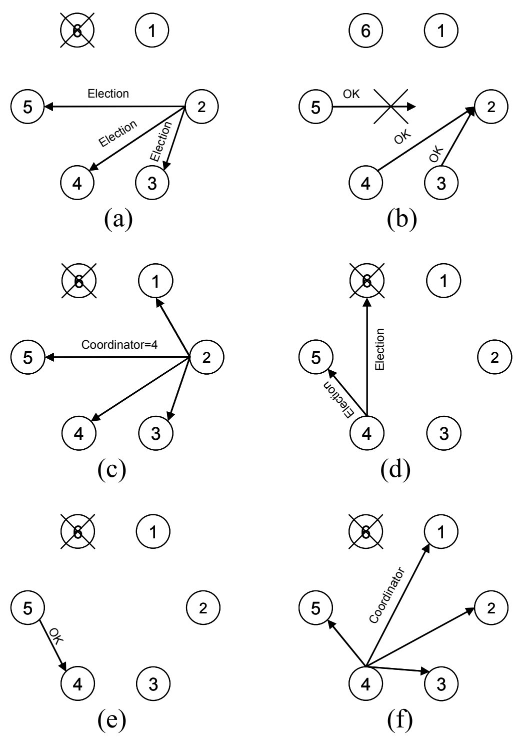


Figure 1: Process 2 notices the coordinator has crashed so sends an election message to processes 3, 4, 5(a), and receives OK message from processes 3, 4 it means that Ok message of process 5 is failed (b), then process 4 is selected as a coordinator (c), and process 4 according to algorithm sends an election message and selects process 5 as a coordinator (d,e,f).

2. Modified Bully Algorithm with Fault Tolerant Mechanism

As it has been mentioned, in Bully algorithm the number of messages that should be exchanged between processes is very high. By presenting the new approach with sort mechanism we decrease the number of messages. Bully with the sort mechanism has a little message passing, but it may consume more time in contrast with Bully Algorithm to find the leader, we describe another approach to modify Bully. In this Algorithm when process P notices that the leader has crashed, it sends an election message to all processes with higher ID number.

Each process that receives election message sends its ID as a respond to process P. If no process responses to process P, it will broadcast one coordinator message to all processes. If some processes response to process P, it will select the process with the highest ID number as coordinator and that will send a new message with selected ID number to all processes. In this manner all processes know the new leader. This approach is a suitable way to select the leader, but when the process with the highest ID number is sent, its message to process P maybe lost. So we want to present a fault tolerant mechanism to prevent this fault. To do this, when process P selected the highest ID number it sends the selected ID to all processes, now the new leader sends election message to processes with greater ID number to be sure that there is no process with great ID numbers.

If a message is received from processes with great ID numbers, it introduces the greatest one as leader. Otherwise it remains the leader again.

3. Modified Ring Algorithm

The methods presented in previous section were about the methods to choose the leader. In this section, we want to introduce an appropriate method to modify Ring algorithm. In this method, the number of message passing in Ring decreases and it prevents sending additional messages to selected leader.

As it is seen in Figure 2, when a process notices that the leader has crashed, it starts sending its ID number in Ring. So it is not necessary for all processes to start sending their own IDs in the Ring. The ID number sent by it reaches to neighbor in the Ring. At this moment the receiving process compares the received ID with its own, and sends whichever is the greatest. This comparison is done by all the processes in the Ring, so only the greatest ID remains in the Ring, and ultimately the ID returns back to its sender. If the ID equals with process ID, the leader becomes known and sends the coordinator message to the Ring.

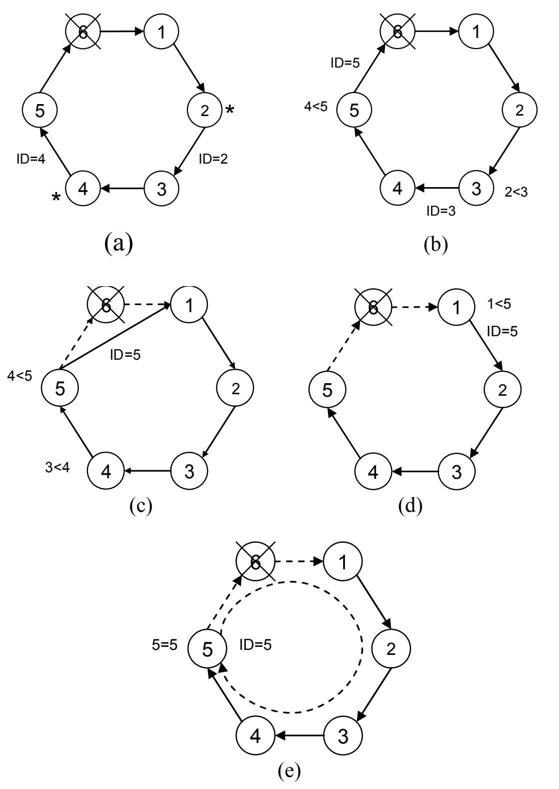


Figure 2: Processes 2, 4 notice that coordinator has crashed concurrently (a), so they send their ID in the Ring (b), and according to algorithm, greatest ID remains in the Ring and its sender becomes known as a coordinator (c, d, e).

It is seen that the above method decreases the overhead sent message greatly. So, if a lot of processes notice the leader has crashed, only the message of the process with the greatest ID number turns around in the Ring and sending smaller ID numbers in the Ring is prevented. This means that if the message with the lower ID reaches to the process which has noticed the leader has crashed, the received message becomes redundant and after this no message is sent there. In implementation the flags can be used to recognize the processes which have noticed the death of the leader. Figure 2 shows this Ring algorithm.

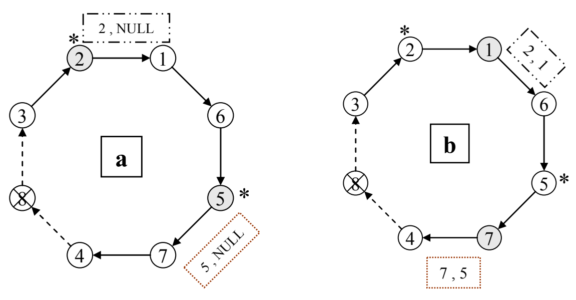
4. Modified Ring with Fault Tolerant

After describing the Modified Ring algorithm in this section we would present this algorithm, so we explain new method to fault tolerant the Modified Ring.

In this method we use another process ID beside the leader ID's. It means that in this method every processor send two IDs to its neighbor. One ID determines the leader in the ring and other ID is spare and it is our leader surrogate. In this method when a process notices that the leader has crashed it select the surrogate instead of old leader and it will become leader. So by doing this method when in first time the leader crashed the processors do not need to determine the leader because the leader has been selected before.

As it is seen in Figure 3, when a process notices that the leader has crashed, it starts sending its ID number and one null number in Ring. So it is not necessary for all processes to start sending their own IDs in the Ring. The ID number sent by it reaches to neighbor in the Ring. At this moment the receiving process compares the received IDs with its own, and sends two greatest numbers. It means that processors sent the next greatest ID beside the main greatest ID that we call it surrogate. This comparison is done by all the processes in the Ring, so the two greatest ID remains in the Ring, and ultimately the ID returns back to its sender. If the ID equals with process ID, the leader becomes known and sends the coordinator message to the Ring. In this method the processes which have noticed the death of the leader send the coordinator message to the Ring and in this packet two ID has been determined. We can use this method in impermanent environments.

In implementation the flags can be used to recognize the processes which have noticed the death of the leader. Figure 3 shows this method. As you see in figure, processes 2 and 5 have noticed the death of the leader so the sent the packet with two IDs one of them are their own ID and the other is null. Other processes put the second greatest ID instead of the null number and ultimately the leader and its surrogate will be determined.



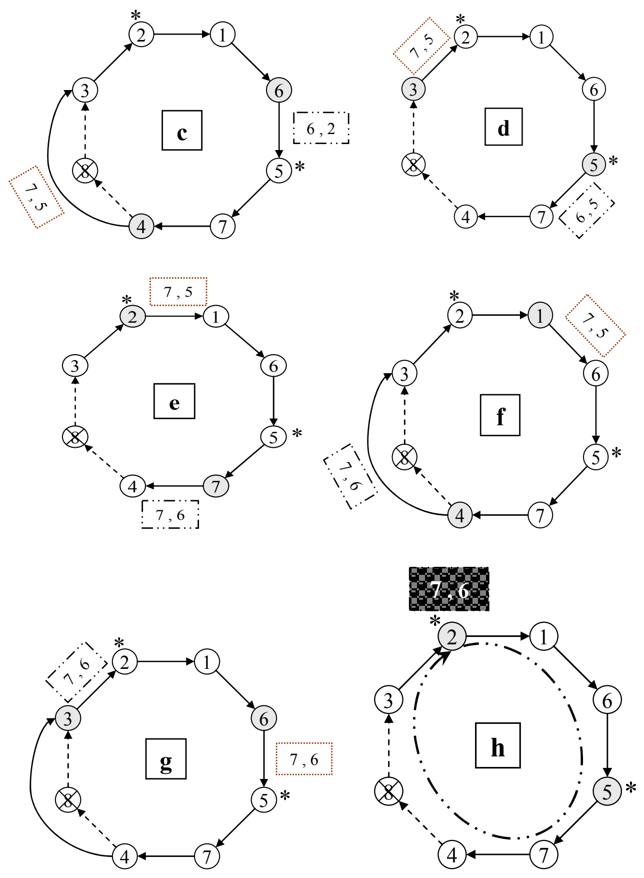


Figure 3: Processes 2, 5 notice that coordinator has crashed concurrently, so they send their packet to neighbor, these packets turn in the ring and surrogate ID add to these packets. Ultimately the leader and surrogate are selected.

5. Evaluation Result

After describing the represented algorithm, in this section we will compare and evaluate the gained results. We will also compare the complexity of message passing between the algorithms and show the improvement of them.

5.1. Analytical Comparison of Modified Bully with Fault Tolerant Mechanism

In Modified Bully with Fault Tolerant Mechanism we have previous parameters, so if only one process detects crashed coordinator we have:

$$\begin{aligned} N_{(i)} &= (n-i)+(n-i)+(n-2) \\ &= 2(n-i)+(n-2) \end{aligned} \qquad (1)$$

Which has Order $O(n)$. When we use the fault tolerant mechanism according to Figure 4 message passing will increase.

$$\begin{aligned} N_{(i)} &= 2(n-i)+(n-2)+2(n-i^{'})+(n-2) \\ &= 2\left[(n-i)+(n-2)+(n-i^{'})\right] \end{aligned} \qquad (2)$$

In which i' is a selected leader ID number in first step. The Order of this method is $O(n^2)$. Figure 4 shows the comparison between Bully Algorithm and modified Bully with fault tolerant mechanism. Figure 5 shows the fault number between modified Bully and modified Bully with fault tolerant mechanism.

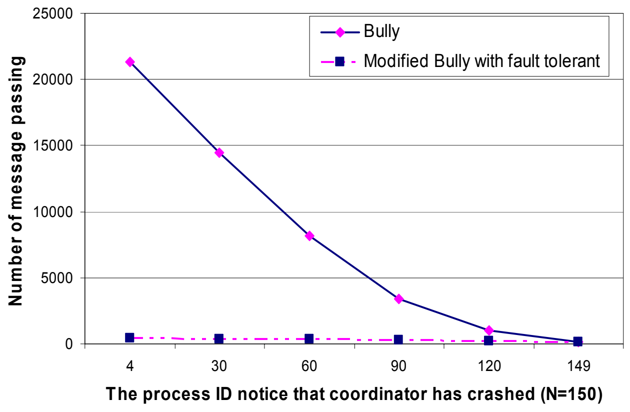


Figure 4: The comparison of message passing in Bully and Modified Bully with fault tolerant (If only one process notices that the coordinator has crashed (N=150)).

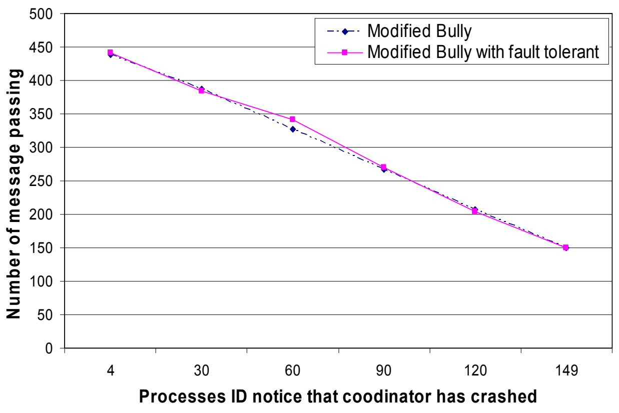


Figure 5: The comparison of message passing between Modified Bully and Modified Bully with fault tolerant mechanism.

Table 1 indicates the number of fault, sent and received message in modified Bully with fault tolerant mechanism. For example, if the 4-th process found that the coordinator has crashed, it will send 145 messages to the processes with larger IDs. But, because of the

Table 1: The number of message passing in Modified Bully with fault tolerant mechanism

Process ID	Number of Message in Bully with fault tolerant	Sent messages	Received messages	Number of fault	message to process with larger ID	Received message from process with larger ID	Coordinator messages
4	442	145	133	12	7	7	149
30	385	119	110	9	3	3	149
60	341	89	32	57	35	35	149
90	270	59	51	8	5	5	149
120	205	29	20	9	3	3	149
149	150	1	0	0	0	0	149

occurrence of the fault in the network, it is impossible to receive all the messages, therefore, it will receive 133 messages. So, 12 faults have occurred. After the primary determination of the coordinator by a message the selected coordinator again sends messages to processes with greater ID than itself and this number is produced at random and it is 7. It also receives 7 messages from these processes and finally the main coordinator will be selected and all the processes receive its ID.

5.2.Modified Ring Algorithm

In this section we will exam modified Ring Algorithm at first we compare the message complexity of Ring Algorithm with modified Ring. Assume that the set of processes in $S=\{i_1,i_2,..i_m\}$ from the processes that find out the crashed coordinator concurrently so the total message passing is:

$$T = n_{\{i_1,i_2,...i_m\}} \times n \qquad (3)$$

$n_{\{i_1,i_2,...i_m\}}$ is the number of processes detecting crashed coordinator and n is the number of processes in the Ring. As you see it Order is $O(n^2)$. The complexity of message passing in modified Ring Algorithm is:

$$\sum_{i=0}^{n} i = n(n-1)/2 = 1/2(n^2 - n) \qquad (4)$$

Which has Order $O(n^2)$. In fact modified Ring Algorithm reduces the message passing that we obtain it from following formula:

$$\sum_{i=1}^{n} (n-i) = 1/2\left[(n-i)(n-i+1)\right] \qquad (5)$$

So the complexity of modified Ring is much lower than the Ring Algorithm.

Figure 6 shows the comparison between Ring and modified Ring Algorithms. In our simulation we assume that the number of existing processes in the ring is 10 and the topology of the Ring has been products randomly. The number of message passing in different status is shown when several processes notice that coordinator has crashed concurrently.

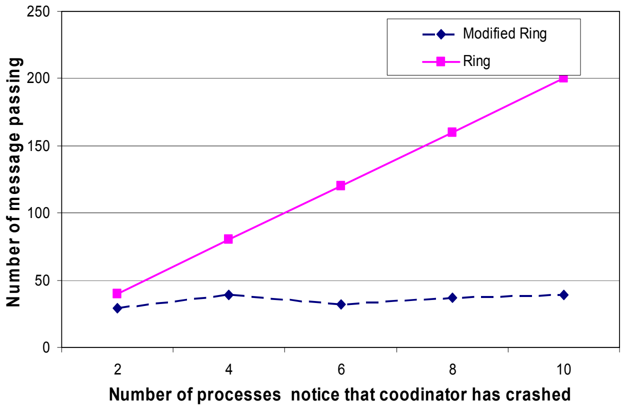


Figure 6: The comparison of message passing in Ring and Modified Ring when several processes concurrently notice that coordinator has crashed.

6. Conclusion and Future Works

Election Algorithms in distributed system play an important role in the system operation. The important algorithms for this kind of work are Bully and Ring Algorithms. The approaches presented in this article have improved these two Algorithms. In modifying Bully Algorithm according to this article; new approaches have been presented.

To improve Bully Algorithm was used simultaneously with fault tolerant mechanism. This algorithm reduces the number of message passing to select the leader, and we compare it with Bully Algorithm the improvement of this algorithm was shown in Figure 4. In this paper a new method is also introduced to improve Ring Algorithm. Modified Ring Algorithm is a suitable approach to leader election and it reduces the message complexity when several processes concurently notice that the coordinator has crashed. So the bandwidth and message passing will be improved.

Also we presented modified Ring algorithm with fault tolerant method in section 4.

In future works we will use these algorithms to leader election in Ad hoc and sensor network as a distributed form. It is also possible to use the election algorithms in dynamic environments.

7. References

[1] J. Welch and H. Attiya, Distributed Computing: Fundamentals, Simulations, and Advanced Topic, London, UK: McGraw-Hill Publishing Company, 2001.

[2] Y. Afek and A. Gafni, "Time and message bounds for election in synchronous and asynchronous complete networks," in Proc. 4th Annu. ACM Symp. on Principles of Distributed Computing, Minaki, Canada, Aug. 1985, pp. 186-195.

[3] J. E. Burns, "A formal model for message passing systems," Tech. Rep. TR-91, Indiana University, Sep. 1980.

[4] D. Dolev, M. Klawe, and M. Rodeh, "An O(nlogn) unidirectional distributed algorithm for extrema finding in a circle," Journal of Algorithms, vol. 3, no. 3, pp. 245- 260, Sep. 1982.

[5] G. Fredrickson and N. Lynch, "The impact of synchronous communication on the problem of electing a leader in a ring," in Proc. 16th Annu. ACM Symp. on Theory of Computing, Washington, D.C., 1984, pp. 493-503.

[6] E. Gafni and Y. Afek, "Election and traversal in unidirectional networks," in Proc. 3rd Annu. ACM Symp. on Principles of Distributed Computing, Vancouver, B.C., Canada, Aug. 1984, pp. 190-198.

[7] R. G. Gallager, Choosing a leader in a network, Unpublished memorandum, M.I.T., Cambridge, Mass., 1977.

[8] R. G. Gallager, P. M. Humblet, and P. M. Spira, "A distributed algorithm for minimum-weight spanning trees," ACM Trans. Program. Lang. Syst., vol. 5, no. 1, pp. 66-77, Jan. 1983.

[9] P. Humblet, "Selecting a leader in a clique in O(n log n) messages," in Intern. Memo., Laboratory for Information and Decision Systems, M.I.T., Cambridge, Mass., 1984.

[10] E. Korach, S. Moran, and S. Zaks, "Tight lower and upper bounds for some distributed algorithms for a complete network of processors," in Proc. 3rd Annu. ACM Symp. on Principles of Distributed Computing, Vancouver, B.C., Canada, pp. 199-207, Aug. 1984.

[11] I. Lavalléé and G. Roucairol, "A fully distributed (minimal) spanning tree algorithm," Information Processing Letters, 23, pp. 55-62, Aug. 1986.

[12] P. M. B. Vitanyi, "Distributed election in an Archimedean ring of processors," in Proc. 16th Annu. ACM Symp. on Theory of Computing, Washington, D.C., pp. 542-547, 1984.

[13] E. Chang and R. Roberts, "An improved algorithm for decentralized extrema-finding in circular configurations of processes," Communications of the ACM}, pp. 281-283, 22, 5, 1979.

[14] W. R. Franklin, "On an improved algorithm for decentralized extrema finding in circular configurations of processors," Communication of the ACM, pp. 336-337, 25, 1982.

[15] G. LeLann, Distributed systems - towards a formal approach, Information Processing Letters, pp. 155-160, 1977.

[16] G. L. Peterson, "An O(nlogn) unidirectional algorithm for the circular extrema problem," ACM Trans. Program. Lang. Syst. 758-762, 4,4 (Oct. 1982).

[17] N. Malpani, J. Welch and N. Vaidya, "Leader Election Algorithms for Mobile Ad Hoc Networks," In Fourth International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, Boston, MA, August 2000.

[18] P. Basu, N. Khan and T. Little, "A Mobility based metric for clustering in mobile ad hoc networks," In International Workshop on Wireless Networks and Mobile Computing, Apr. 2001.

[19] R. Gallager, P. Humblet and P. Spira, "A Distributed Algorithm for MinimumWeight Spanning Trees," In ACM Transactions on Programming Languages and Systems, vol.4, no.1, pages 66-77, January 1983.