International Journal of Current Advanced Research
ISSN: O: 2319-6475, ISSN: P: 2319-6505, Impact Factor: 6.614
Available Online at www.journalijcar.org
Volume 7; Issue 3(L); March 2018; Page No. 11260-11264
DOI: http://dx.doi.org/10.24327/ijcar.2018.11264.1946

Research Article

AN INTELLIGENT FAULT TOLERANCE ALGORITHM FOR ELECTING COORDINATOR IN COORDINATED CHECKPOINTING WITH DUAL COORDINATOR AND SMART INTERVAL

Manoj Kumar Niranjan[1] and Mahesh Motwani[2]

[1]Rustamji Institute of Technology, BSF Academy, Tekanpur
[2]UIT-RGPV, Bhopal

A R T I C L E I N F O

Article History:

Received 14th December, 2017
Received in revised form 10th January, 2018 Accepted 20th February, 2018
Published online 28th March, 2018

Key words:

Distributed Systems, Checkpointing, Fault Tolerance, Smart Interval.

A B S T R A C T

Checkpointing is a well known technique that tolerates the transient faults. The developed algorithm selects a new coordinator efficiently in case of failure of coordinator. It communicates the messages within a specified time interval only and tolerates the fault using dual coordinator methodology.

INTRODUCTION

A distributed system is an application that executes a collection of protocols to coordinate the actions of multiple processes on a network to perform a single or small set of related tasks. A distributed system is called Fault Tolerant if it can recover from failures without performing incorrect actions. The failure may be a network failure, network partition failure, timing failure, byzantine failure, omission failure, fail-stop failure or halting failure [1]. The fault tolerance can be achieved by using Checkpointing which is a very well known technique of fault tolerance. Our paper presents a new algorithm for Checkpointing which can tolerate the failure of any process (node) as well as Coordinator Process (Node) by using dual coordinator methodology. In case of failure of coordinator, our algorithm selects new coordinator efficiently.

Checkpointing

Checkpointing is the method of periodically recording the states of the system onto the stable storage. Any such periodically saved state is called the checkpoint of the process [2]. A global state [3] of a distributed system is a set of individual process state per process [2]. Checkpointing may be one of two types, i.e., independent and coordinated Checkpointing. In Independent Checkpointing, each process takes checkpoint independently without requiring any synchronization when a checkpoint is taken [4].

Corresponding author:* **Manoj Kumar Niranjan
Rustamji Institute of Technology, BSF Academy, Tekanpur

In coordinated Checkpointing, the processes coordinate their Checkpointing action in such a way that the set of local checkpoints taken is consistent [5,6,7].

Existing Work

In the existing work, the communication is initiated by the coordinator with other processes to create a checkpoint. If message communication takes place after checkpoint request of coordinator, the global checkpoint may be inconsistent shown in fig. 1 in which process P_0 sends message m after receiving a checkpoint request from the coordinator. If process P_1 receives message m before the checkpoint request, the checkpoint will become inconsistent because checkpoint $c_{1,x}$ confirms that message m is received from P_0, while checkpoint $c_{0,x}$ says that it is not sent from P_0. [8]

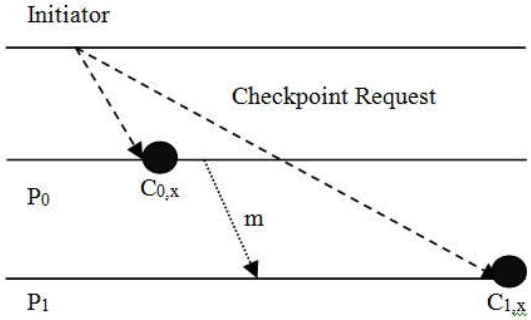

Fig 1 Message communication between P_0 and P_1 causing inconsistent checkpoint

In another protocol, the message communication is allowed within a fixed time interval only. This concept diminishes

message communication [9] which is beneficial in decreasing the communication overhead. The main drawback of this protocol is that it does not tolerate the fault in case of failure of coordinator.

Another Checkpointing protocol specifies that, the coordinator process is not fixed which reduces the probability of failure of coordinator but the message communication could be accomplished at any time i.e., there is no concept of fixed time interval for message communication. Hence it increases communication overhead [10].

In another protocol, all processes take checkpoints at the end of their respective smart interval to form a global consistent checkpoint. Since the Checkpointing is allowed only within smart interval, the protocol will minimize various overheads like Checkpointing overhead, message logging overhead etc [12].

Proposed Work

The developed algorithm presents a new method that not only selects the new coordinator in case of failure of coordinator but also tolerates the fault. Each process knows the priority number of rest of the processes. As soon as a process knows that the coordinator has failed, it sends the message to the process with second highest priority to be the new coordinator. On receiving this message, the new coordinator sends messages to all the remaining processes that I am the new coordinator. Results shown in Table-2 show that the algorithm developed by us takes less time in selecting the coordinator as compared to Modified Bully Algorithm [11]. The better coordinator selection time will provide faster execution of the processes.

System Model

Let us consider a distributed system of 'n' processes, P_0, P_1, ……, P_{n-1}. The no. of processes 'n' is fixed for the duration of execution. Let the checkpoints be denoted as CP^k_i, (here k is the process no. and i the checkpoint no.). The initial checkpoint is taken when the system is being initialized. We are assuming followings:

1. The network is secure, reliable and homogeneous with infinite bandwidth and zero latency. The topology doesn't change and the transport cost is zero.
2. The network guarantees reliable FIFO (First In First Out) delivery of messages between any pair of processes. The assumption of FIFO delivery assures the message synchronization.
3. There is one initiator process. In case of failure of initiator process, a new process will act as initiator.
4. The message communication will take place only in specified time interval which is elapsed between the control messages for prepare checkpoint and take checkpoint. If any process sends a message within this time interval, it has to be logged and the process execution is continued. This enables handling of lost messages.

Protocol Description

The checkpoint initiator process sends checkpoint-prepare-request-message to other processes to start Checkpointing. The other processes send their responses to the initiator process. If initiator process received replies from all processes within specified time-interval then it sends take-checkpoint-request-message and if initiator process does not receive replies from any process within specified time-interval then it will send abort-checkpoint-request-message. The set of checkpoint of all processes received by initiator process is called global checkpoint. The i^{th} global checkpoint is the set $CP_i=\{CP^0_i, CP^1_i,\ldots\ldots, CP^{n-1}_i\}$ in a system of n processes.

The maximum transmission delay to reach a message to destination is t. The T is the Checkpointing interval. Here T>3t, since checkpoint interval (T) is obviously greater than specified time-interval and the length of specified time-interval is bound to be at least 3t to survive the transmission delay of control messages.

Now, if the initiator process fails, a new initiator process has to be selected. The protocol should also save the global checkpoint which is stored at the initiator. Our protocol creates a backup copy of global checkpoint which can be used at the failure of initiator process. The backup copy will be stored at the process which will act as initiator, if initiator process fails. To select new initiator, one of three algorithms, i.e., Bully algorithm, Chang and Roberts algorithm or the modified Bully algorithm (proposed by me and described as procSelCoord) may be used. Here I am using my algorithm, i.e., procSelCoord.

Checkpointing Process

The checkpoint process starts at the time of system initialization. After T time interval of previous checkpoint, the initiator process starts the process of Checkpointing. The first initiator P_{init} and backup initiator P_{binit} will be selected by leader-election algorithm suggested by Gallager, Humblet and Spira.

The initiator process P_{init} sends checkpoint-prepare-request-message to all other processes at t_{prep}. On receiving checkpoint-prepare-request-message, each process write tentative checkpoint after sending response to the initiator.

1. Now, if initiator receives response from all processes, within ($t_{prep}+2*T_{trns}$), the initiator process sends *take-checkpoint-request-message* to all processes. When receiver receives *take-checkpoint-request-message* from initiator process, the *tentative checkpoint* is made *permanent*. This will save the states of all processes which are responsible for preparing a global checkpoint.
2. Now, suppose if one or more process fails after responding to checkpoint-prepare-request-message, then the tentative checkpoint is used to recover the failed process.
3. Now suppose if one or more process fails to respond to checkpoint-prepare-request-message, the initiator process sends abort-checkpoint-request-message to all processes. On receiving this, the tentative checkpoint is deleted. The copy of unacknowledged message keeps in a log in this case.
4. If the global checkpoint created successfully, then it has to be saved on backup initiator P_{binit}. The P_{init} sends the global checkpoint data to P_{binit}. After receiving the global checkpoint P_{binit} sends acknowledgement message to P_{init}. After receiving the acknowledgement message from P_{binit}, P_{init} starts the process of next checkpoint.
5. If the P_{init} fails, then there may be three states. P_{init} may be fail before starting the checkpoint process, after

starting the checkpoint process but before completion of checkpoint process or after completion of checkpoint process, but before sending the global checkpoint to backup initiator.

6. If P_{init} fails before starting the checkpoint process, then P_{binit} and other process will not get checkpoint-prepare-request-message from P_{init}. If P_{binit} does not receive the checkpoint-prepare-request-message within the specific time interval, then it first sends a test message to P_{init} to confirm the status of initiator. If P_{init} replies positively, then P_{binit} takes no action, otherwise P_{binit} starts the process of next checkpoint. It also resets its role, now, it acts as initiator and runs leader-election algorithm proposed by me (procSelCoord) to find the next initiator. After finding the next initiator (which will act as backup initiator), the Checkpointing process continues as above.
7. If P_{init} fails after starting the checkpoint process but before completion of checkpoint process, then P_{binit} will not get global checkpoint data. If P_{binit} does not get the global checkpoint data which should be received within $(t_{prep}+2*T_{trns})$, then it sends a test message to P_{init} to confirm the status of initiator. If P_{init} replies positively, then P_{binit} takes no action, otherwise P_{binit} starts the process of next checkpoint. It also resets its role, now, it acts as initiator and runs coordinator-election algorithm, proposed by me (procSelCoord), to find the next initiator. After finding the next initiator (which will act as backup initiator), the Checkpointing process continues as above.
8. If P_{init} fails after creating the global checkpoint but before sending it to backup initiator, then also like previous step, backup initiator P_{binit} will not receive the global checkpoint data within $(t_{prep}+2*T_{trns})$. Now, it will send a test message to P_{init} to confirm the status of initiator. If P_{init} replies positively, then P_{binit} takes no action, otherwise P_{binit} starts the process of next checkpoint. It also resets its role, now, it acts as initiator and runs coordinator-election algorithm, proposed by me (procSelCoord), to find the next initiator. After finding the next initiator (which will act as backup initiator), the checkpointing process continues as above.
9. In step (7) and (8), if P_{binit} gets positive reply from P_{init}, but does not receive the global checkpoint data, then it sends request message to send the global checkpoint data, i.e., send-global-checkpoint-message. It waits for t time to receive the global checkpoint data. If it does not receive the global checkpoint within t, then it again sends test message to Pinit and if it gets positive reply then it repeat the step (9) until it get the global checkpoint data. If it does not get positive reply, then it starts acting as initiator like step (7) and (8).

Algorithm

Step-I

This step is executed to select the initiator process P_{init} and backup initiator P_{binit}

1. Execute the coordinator-election algorithm described as procSelCoord on the set of all process {Pi: 1≤i≤n}
2. Find the best suitable process using procSelCoord and make it initiator P_{init}
3. Exclude the P_{init} from the set of all processes and run the leader election algorithm procSelCoord on this new set, i.e., {Pi: 1≤i≤n and i≠init)
4. Find the best suitable process for initiator using procSelCoord and make it backup initiator P_{binit}.

Step-II

This step is executed at initiator process P_{init}

1. Send *checkpoint-prepare-request-message* to remaining processes at t_{prep} for $(k+1)^{th}$ checkpoint
2. Remove $(k-1)^{th}$ checkpoint, if exist.
3. Receive response from other processes within $(t_{prep}+2*T_{trns})$
4. If all processes respond positively then Send *take-checkpoint-request-message* to all processes.
   Create *global-checkpoint* and send it to P_{binit}.
   Else (if even a single process does not respond positively or response does not arrive to initiator process)

a) Send *abort-checkpoint-request-message* to all processes
b) Retain copies of unacknowledged messages in a log

Step-III

This step is executed at other process P_{oth}

1. Receive *checkpoint-prepare-request-message* from initiator at t_{rec}
2. Send own response to initiator
3. If response is positive then Call save_state(P_{oth}) to write *tentative-checkpoint* asynchronously
4. Wait for decision of P_i till $(t_{rec}+T_{trns}+T_{trns})$
5. If received decision is *take-checkpoint-request-message* then Change status of *tentative-checkpoint* to permanent
   Else Delete *tentative-checkpoint*
6. Delete messages whose acknowledgements have received. Log unacknowledged messages.

Step-IV

This step is executed at any process P_{any} for receiving message

1. If ((checkpoint number in message)=(checkpoint number in P_{any}))
a. Send (tag1,s_id)
b. Receive(message)
2. else if ((checkpoint number in message)>(checkpoint number in P_{any}))
a. save_state(P_{any})
b. send(tag1,s_id)
c. receive(message)
3. else if ((checkpoint number in message)<(checkpoint number in P_{any}))
a. send (tag2,s_id)
b. receive(message)

Step-V

This steps is executed at any process P_{any} for writing unacknowledged messages

i. for all k
if (ack[k]=0) then write k^{th} message in buffer

Step-VI

This steps is executed at backup initiator process P_{binit} for writing unacknowledged messages

for all k

if (ack[k]=0) then write k^{th} message in buffer

Sub Algorithm (procSelCoord)

The algorithm to elect leader in case of failure of coordinator

Step-I

Any non-initiator process executes this step

a) Smart Interval Started//Start smart interval
b) If *checkpoint-prepare-request-message received* from initiator Then
   Prepare the Checkpoint accordingly and Exit
   Else
   if *no-message-received* AND *smart-interval-ended*
   Then go to Step C
c. Send message to process with PRIORITY=(HIGHEST PRIORITY-1)
d. Update the initiator priority, i.e., HIGHEST PRIORITY=HIGEST PRIORITY-1

Step-II

This step is executed at process with PRIORITY=(HIGHEST PRIORITY-1)

1. Received message NEW-LEADER from any process
2. Update initiator priority=MYSELF
3. Send message to all remaining processes with HIGHEST PRIORITY=HIGHEST PRIORITY-1

Performance Results

The presented algorithm is simulated using parallel virtual machine java libraries. The environment used for simulation of algorithm is Windows 10 with JDK8 and for sub-algorithm (procSelCoord) is Ubuntu 13.10 with Open JDK 7. Since, we assumed consistent network bandwidth, we created all the process on a single computer with Intel i3 processor and 2GB DDR3 RAM. The results of simulation are as under:

Table 1 Results of algorithm

No. of Processes	Total Execution Time(milli seconds)	Time to Checkpoint (milli seconds)	No. of Failures in Coordinator	No. of Failures in non-Coordinator	Total Execution Time (milli seconds)
5	1000	100	1	2	1680
6	1100	110	2	4	2024
7	1200	120	3	6	2227
8	1300	130	4	8	2027
9	1400	140	5	10	1766
10	1500	150	6	12	2157
11	1600	160	7	14	2438
12	1700	170	8	16	2360
13	1800	180	9	18	2586
14	1900	190	10	20	2422
15	2000	200	11	22	2407
16	2100	210	12	24	2703
17	2200	220	13	26	3227
18	2300	230	14	28	3008
19	2400	240	15	30	3056
20	2500	250	16	32	3063
21	2600	260	17	34	3211
22	2700	270	18	36	3422
23	2800	280	19	38	3602
24	2900	290	20	40	4087
25	3000	300	21	42	4321
26	3100	310	22	44	3829
27	3200	320	23	46	3922
28	3300	330	24	48	3781
29	3400	340	25	50	3782
30	3500	350	26	52	4235
31	3600	360	27	54	4376
32	3700	370	28	56	4399
33	3800	380	29	58	4577
34	3900	390	30	60	4325
35	4000	400	31	62	4761
36	4100	410	32	64	4914
37	4200	420	33	66	4930
38	4300	430	34	68	4813
39	4400	440	35	70	4985
40	4500	450	36	72	5305
41	4600	460	37	74	5156
42	4700	470	38	76	5329
43	4800	480	39	78	5329
44	4900	490	40	80	6118
45	5000	500	41	82	5501
46	5100	510	42	84	6009
47	5200	520	43	86	6157
48	5300	530	44	88	5860
49	5400	540	45	90	6360
50	5500	550	46	92	6196
51	5600	560	47	94	6219
52	5700	570	48	96	6399
53	5800	580	49	98	6563
54	5900	590	50	100	6946

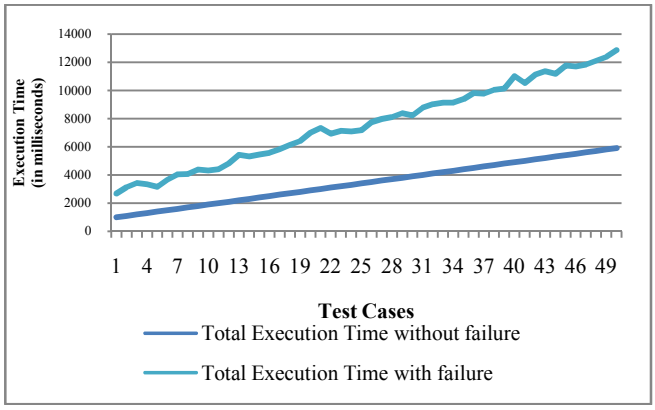

Fig 2 Graphical representation of results

Table 2 Results of sub-algorithm (procSelCoord)

Test Case	Existing Algorithm (Leader Election Time in Nanoseconds)	New Algorithm (Leader Election Time in Nanoseconds)	Difference (Leader Election Time in Nanoseconds)
1	1266498203	991008242	-275489961
2	1281315083	983011291	-298303792
3	861042068	973207262	112165194
4	876048862	861143185	-14905677
5	891176278	851309321	-39866957
6	906080737	841284902	-64795835
7	770943876	731246367	-39697509
8	1251484351	929271510	-322212841
9	785942972	710347724	-75595248
10	665670598	601252026	-64418572
11	1416515141	891265461	-525249680
12	1431641595	880390275	-551251320
13	1011235127	870433866	-140801261
14	1027204968	861219791	-165985177
15	1041107883	851172916	-189934967
16	1056340522	840479304	-215861218
17	921106138	836181520	-84924618
18	1401693770	821176971	-580516799
19	936087268	811181424	-124905844
20	681074417	600374247	-80700170
21	696678865	691711953	-4966912
22	635992439	681144737	45152298

23	651030672	641096579	-9934093
24	732075833	661096860	-70978973
25	741028774	730134095	-10894679
26	620996552	620272001	-724551
27	710971555	710245978	-725577
28	667021367	620436496	-46584871
29	755366375	710394753	-44971622
30	1506920660	700233351	-806687309
31	719963377	690223047	-29740330
32	735614663	680160132	-55454531
33	981296926	670106840	-311190086
34	995512943	660188926	-335324017
35	951615687	650185359	-301430328
36	442128932	440158053	-1970879
37	966358461	630110535	-336247926
38	650585207	620154767	-30430440
39	625075100	610152484	-14922616
40	655841932	600087002	-55754930
41	1116641987	590104607	-526537380
42	1132469713	580076018	-552393695
43	585100995	570056730	-15044265
44	600963689	560106417	-40857272
45	575161973	550080393	-25081580
46	590187053	541311592	-48875461
47	570891393	530775815	-40115578
48	1146466624	520049481	-626417143
49	1102384906	510743337	-591641569
50	606057126	500051328	-106005798

CONCLUSION

In our algorithm, whenever initiator process Pi sends checkpoint-prepare-request-message for (k+1)th checkpoint, the protocol will automatically delete the (k-1)th global checkpoint which results simplified garbage collection. The results shown in Table 1 of new algorithm developed by us show that processes continue their execution in presence of faults in coordinator and non-coordinator processes. Further our algorithm takes less time in electing the coordinator as compared to modified bully algorithm in case the coordinator process fails.

References

1. Introduction to Distributed System Design, Google Code University, http://code.google.com/edu/parallel/dsd-tutorial.html#Basics
2. D. Manivannan, R.H.B. Netzer & M. Singhal, "Finding Consistent Global Checkpoints in a Distributed Computation", IEEE Trans. On Parallel & Distributed Systems, Vol.8, No.6, pp. 623-627 (June 1997)
3. J. Tsai & S. Kuo, "Theoretical Analysis for Communication-Induced Checkpointing Protocols with Rollback-Dependency Trackability"; IEEE Trans. On Parallel & Distributed Systems, Vol.9, No. 10, pp. 963-971 (October 1998)
4. B. Bhargava and S.R. Lian, "Independent Checkpointing and Concurrent Rollback for Recovery in Distributed Systems-An Optimistic Approach", Proceeding of IEEE Symposium on Reliable Distributed Systems, pp. 3-12 (1988)
5. Guohong Cao, and Mukesh Singhal, "On Coordinated Checkpointing in Distributed Systems," IEEE Transactions On Parallel And Distributed Systems," Vol. 9, No. 12, pp.1213-122 (Dec.1998)
6. Sharma D. D. and Pradhan D. K., "An Efficient Coordinated Checkpointing Scheme for Multicomputers," Proc. IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems, pp 36-42 (June 1994)
7. E.N. Elnozahy, D.B. Johnson, and W. Zwaenepoel, "The Performance of Consistent Checkpointing," Proc. 11th Symp. Reliable Distributed Systems, pp. 39-47 (Oct. 1992)
8. E.N. (Mootaz) Elnozahy, Lorenzo Alvisi, Yi-Min Wang and David B. Johnson, "A Survey of Rollback-Recovery Protocols in Message-Passing Systems", ACM Computing Surveys (CSUR), Volume 34, Issue 3 (September 2002) Page(s):375-408 (2002)
9. Ch. D.V. Subba Rao and M.M. Naidu, "A New, Efficient Coordinated Checkpointing Protocol Combined with Selective Sender-Based Message Logging", IEEE/ACS International Conference on Computer Systems and Applications, AICCSA 2008, pp. 444-447 (2008)
10. Sarmistha Neogy, Anupam Sinha, Pradip K Das, "CCUML: A Checkpointing Protocol for Distributed System Processes", IEEE Transactions on TENCON 2004, IEEE Region 10 Conference, Volume B, 21-24 Nov. 2004, Page(s):553-556 (2004)
11. Sandipan Basu, "An Efficient Approach of Election Algorithm in Distributed Systems", Indian Journal of Computer Science and Engineering (IJCSE), vol. 2, No. 1, pp. 16 -21. March 2011
12. J. Makhijani, M.K. Niranjan, M.Motwani, A.K. Sachan, A. Rajput, "An efficient protocol using smart interval for coordinated checkpointing", International Conference on Advances in Information Technology and Mobile Communication-AIM 2011

How to cite this article:

Manoj Kumar Niranjan and Mahesh Motwani (2018) 'An Intelligent Fault Tolerance Algorithm For Electing Coordinator In Coordinated Checkpointing With Dual Coordinator And Smart Interval', *International Journal of Current Advanced Research,* 07(3), pp. 11260-11264. DOI: http://dx.doi.org/10.24327/ijcar.2018.11264.1946

****