Fault Tolerance Multi Agent Co-Ordination: A Petri Net based Approach

Sudipta Acharya
Dept. of Information technology
National Institute of Technology
Durgapur, India

Prajna Devi Upadhyay
Dept. of Information technology
National Institute of Technology
Durgapur, India

Animesh Dutta
Dept. of Information technology
National Institute of Technology
Durgapur, India

ABSTRACT
As technology shifts from centralized computing to distributed computing and then to ubiquitous computing, the users are more dependent on the computer system for task delegation. Here autonomous agent and Multi Agent System (MAS) plays an important role to perform the task delegated by the user. As the fault in MAS is not-deterministic in nature, so designing fault tolerant MAS is a challenging research area. Here we propose a dynamic fault tolerant MAS interaction protocol. We model the proposed protocol using a high level Petri net. The model is analyzed to check the fault tolerance capability in different fault tolerant situation of the system.

General Terms
Distributed computing.

Keywords
Agent, Multi Agent System (MAS), Petri Nets, Fault tolerance.

1. INTRODUCTION

1.1 Agent and Multi Agent system(MAS)
An agent is a computer system or software that can act autonomously in any environment,makes its own decisions about what activities to do, when to do, what type of information should be communicated and to whom, and how to assimilate the information received. Multi-agent systems (MAS)[1,2] are computational systems in which two or more agents interact or work together to perform a set of tasks or to satisfy a set of goals. Each agent of system has following information attached with it.

<AID,{capability set},TID,state>

Where AID = Agent identification number (each agent has unique ID), {Capability set} = consists set of tasks which can be performed by this agent, TID = Identification number of task which the agent is currently performing, State =represents the state of system while performing task. During the execution system goes through a number of states.

1.1.1 Peer Agent
Let a agent a_m is performing a task t_m ,then another agent a_n will be reffered as peer agent of a_m if t_m Є {capability set of a_n }.For example Let capability set of a_i = $\{t_1,t_2\}= T_i$, for a_j agent = $\{t_1,t_3\}=T_j$, for a_k agent = $\{t_2,t_4\}= T_k$, where $\{a_i,a_j,a_k\}$Є A and $\{t_1,t_2,t_3,t_4\}$ Є T.where A=set of all agents in the system.T=set of all tasks in the system.while a_i is performing t_1 then according to the definition of peer agent its peer agent will be a_j as t_1 Є T_j.similarly while a_i is performing t_2 then its peer agent will be a_k as t_2 Є T_k .

1.2 Petri Nets for Agent coordination
Petri Nets [3, 4] were first conceptualized by Carl Adam Petri in 1962. Petri nets and Color Petri Nets are graphical tools for the formal description of systems whose dynamics are characterized by concurrency, synchronization, and mutual exclusion, which are typical features of distributed environment. Petri Nets have been widely used to describe the Multi Agent Systems for a long time. Color Petri Nets have been used in [5] to achieve agent scheduling in open dynamic environments. The representation of composite behaviors through Color Petri Nets has been done in [6].

1.2.1 Reachability in petri-net
Let the initial marking of the Petri net be M_0. A marking M_j is said to be reachable from marking M_i if there exists a sequence of transitions that takes the Petri net from M_i to M_j . If I is the incidence matrix of the model, then the reachability criterion can be specified by the following matrix equation:

$$M_i + I.\ \sigma = M_j \qquad (1)$$

where σ is the sequence of transitions. This is a necessary but not sufficient condition. If the goal state in the Petri-Net is reachable for any fault state, it indicates that goal can be achieved through formal derivation and the petri-net is fault tolerant.

1.3 Fault tolerance
A characteristic feature of distributed system that distinguishes them from single machine systems is the notion of partial failure which may happen when one component in a distributed system fails. This failure may affect the proper operation of other components. An important goal in distributed systems design is to construct the system in such a way that can be automatically recovered from partial failures without seriously affecting the overall performance [7]. In our paper we take a closer look at techniques for making MAS fault tolerant.

1.4 Happens-before relationship
The expression a→b is read "a happens before b" and means that all processes agree that first event a occurs, then afterword event b occurs[7]. The happens-before relation can be observed directly in two situations:

1. If events a and b occur on the same process and the occurrence of event a preceded the occurrence of event b then a→b =TRUE
2. if a is the event of sending a message m in a process and b is the event of receipt of the same message m by another process then a→b is also true. A message cannot be received before it is sent or even at the same time it is sent, since it takes a finite, nonzero amount of time to arrive.

3. Happens-before is a transitive relation i.e. If a→b and b→c then a→c .

2. RELATED WORK

It is very obvious that while performing a task, one or more than one agents can stop executing. Failure of such agents can be of different type. For example Crash type, Byzantine, omission. In our paper we consider Crash type failure of agent, i.e. agent just stops executing or producing output. Till now a number of fault tolerance methods have been proposed. In [8], authors propose a preventive method to achieve fault tolerance by replicating agents. As discussed by [9], software replication in distributed environments has some advantages over other fault tolerance solutions. But in both the papers [8,9] replication methods increase cost of application much more than optimal cost because for every agent replicates are maintained. To overcome this in [10] authors propose a dynamic, automatic plan based replication mechanism to achieve fault tolerance. A new factor "criticality" of agent is proposed here which is calculated according to the difficulties to perform tasks in agents plan graph. After every time interval Δt according to the criticality of agent number of replicated agents are employed for highly critical agents. But this method takes into account the prediction of the future behaviour of the agents and their influence over the other agents of the society which may not be always true.In [11] author proposes a strategy for fault tolerance using sentinels. The sentinel agents listen to all broadcast communications, interact with other agents, and use timers to detect agent crashes and communicate link failure. The main problem within this approach is that sentinels also are subject of faults. Authors in [12] introduce a strategy based on Adaptive Agent Architecture. This strategy uses the teamwork to cover a multi-agent system from broker failures. This approach does not deal completely with agent failures since only some agents (the brokers) or part of them can be replicated. In [13] a strategy is proposed based on 2 phase Decision making for fault handling in MAS. In this strategy faulty agents broadcast or multicast help request, after getting that request helper agents decide whether they are able to help or not, and if yes they determine the number of help requests and if there exist a number of faulty agents seeking help then helper agents decide in which sequence they will help faulty agents. But a problem of this strategy is it is not applicable for crash type of failure, because it is not possible for any agent to request for help after crash of that agent.

3. SCOPE OF WORK

Although there have been a number of contributions in the area of fault tolerance in MAS, most of them only concentrate on replication mechanism of agents and pay limited attention to the ways of detection and handling of faults if they occur. Some deal with predictive mechanisms to handle fault but these mechanisms go wasted if the fault does not occur in the MAS. There is a need for efficient fault detection and resolving mechanism which does not lead to a deadlock even if some of the agents crash and resumes the work of the faulty agent by searching for peer agents from the agent pool. In this paper, we have proposed such a mechanism where the agents performing the tasks share their state information in periodic intervals so that any fault, if occurs, can be detected and handled accordingly. We have also modeled the protocol with the help of color Petri nets and shown the formal proof of the reachability of the goal state of the system. In this way, we verify that the system achieves its goal even if some faults occur in the MAS.

4. MODELLING MAS FAULT TOLERANCE

4.1 Problem definition

Let us define a few terms to understand the problem definition

4.1.1 Interface Agent

In our MAS, an interface agent is one which accepts the user's query and determines the task to be performed from the query. It also divides the task into a number of sub-tasks along with determining their happens before relationships.

4.1.2 Concurrent tasks

For two concurrent tasks t_i, t_j, we write $(\neg(t_i \rightarrow t_j)) \wedge (\neg(t_j \rightarrow t_i)) = \text{TRUE}$

4.1.3 Dependent tasks

If a task t_j is dependent on a task t_i, we write $(t_i \rightarrow t_j) = \text{TRUE}$

4.1.4 Dependency graph

We define a dependency graph to represent the happened before relationship between the subtasks. A dependency graph is a directed graph where each subtask t_i is represented by a vertex - v_i and a dependency relation between two subtasks t_i and t_j is represented by a directed edge from vertex v_i to vertex v_j.

Let a user submit a query Q to a MAS. Initially Q is given to interface agent. This agent finds out the task to be performed from the query. Let this task be called T. T is divided into a number of subtasks. Let those subtasks be t_1, t_2,…..,t_n. There may exist a happened–before relationship between these subtasks. Each subtask is performed by an agent. The subtask starts with an initial state, and during execution it goes through a number of states to reach to a final state indicating that the subtask is complete. After getting required resources, the agent starts execution. During execution if one or more than one agents crash due to some reason then how to tolerate that fault such that user does not come to know about this fault in system and further execution can be carried on from faulty state of task to finish that task successfully as well as all subtasks to reach to the goal state i.e. to satisfy user request, we have to design a protocol to get fault tolerant MAS. Total fault tolerant protocol should be designed in such a way that minimum number of agents should be employed to perform total task.

4.2 Proposed protocol to get fault tolerant MAS

The proposed protocol to get fault tolerance is shown in figure 1.

4.3 Petri net representation of proposed protocol

The Petri net representation of our proposed protocol is shown in figure 2. The Petri net consists of a number of places and transitions. There are two types of places timed place and non-timed place. A transition connected to a timed place can occur only when the time interval of the place elapses. Each place contains a set of markers called color tokens. The Petri-net model of the proposed protocol given in figure has 28 states and 52 transitions. P_1 is the place where all sub-tasks initially reside and P_{15} is the place where each sub-task after completion resides. Arc which has weight other than one is specified explicitly in the petri-net. In order to prove that the goal state is reachable even if any agent faults for any of the five cases described later, we can use the matrix equation for reachability of a marking(section I.C.1*)*The places are described as

P_1: Contains all subtasks after dividing the main task given by the user.

P_2: Contains pool of agents.

P_3:Place of concurrent independent tasks with allocated agents.

P_4: Place of dependant tasks.
P_5: Initially contains those concurrent tasks with allocated agents who get their required resources and are ready to be executed. If more than one token is here, this place contains multiple concurrent tasks with allocated agents.
P_6: Contains those concurrent tasks with allocated agents who have not got required resources, agents are waiting in this place. i.e. not ready to be executed.
P_7: Contains dependant tasks of single running agents in the system which has no concurrent tasks in waiting stage.
P_8: Contains single concurrent task with allocated agent temporarily which is ready to be executed.
P_9: Contains single task with allocated agent which is ready to be executed has no concurrent task in waiting stage but has a one or more dependant tasks.
P_{10}: Contains dependant tasks of singly running agents with allocated peer agents who has no concurrent tasks in waiting stage.
P_{11}: Contains incomplete single task which is aborted due to crash of it's agents during execution in place P_9
P_{12}: Contains same incomplete task of place P_{11}.
P_{13}: Contains dependent tasks of selected peer agent from its capability set which is employed to perform incomplete single task of P_{11}.
P_{14}: Contains single task with allocated agent ready to be executed.
P_{15}: Contains finished tasks with agents which have finished that task successfully.
P_{16}: For singly running agent which have directly dependent tasks this timed place contains concurrent tasks of it with allocated agents which are in waiting stage. Each agent has timer information "t_m".
P_{17}: Contains incomplete single task which is aborted due to fault of the agent executing it in P_{14}.
P_{18}: Contains directly dependent tasks of incomplete task which is in place P_{17}.
P_{19}: Contains directly dependent tasks of the task which is in place P_{17} with allocated peer agents.
P_{20}: Contains dependent tasks of selected peer agent from its capability set which is employed to perform incomplete single task.
P_{21}: Contains multiple concurrent tasks with allocated agents that are ready to be executed, each has timer information "tm", i.e. it is a timed place.
P_{22}: Contains failed tasks which are in place P_{21}.
P_{23}: Contains directly dependent tasks of incomplete tasks which are in place P_{22}.
P_{24}: contains directly dependent tasks which are in P_{23} with allocated peer agents.
P_{25}: for each agent of P_{24} it contains dependent task of selected peer agent from its capability set which are employed to perform their corresponding incomplete tasks.
P_{26}: Contains concurrent tasks with allocated agents which are not ready to be executed of the single running agent in place P_{14} with no dependant tasks. Agents of this place have timer info "t_m", i.e. it is a timed place.
P_{27}: Contains 5 copies of the single task running in the system, which is in P_{14} place, which have no other concurrent tasks as well as dependent tasks.
P_{28}: Contains 5 peer agents for same single task whose copies are in place P_{27}.
The descriptions of transitions are,
t_1: It will be fired for those tasks in P_1 place which are independent of any other tasks of the system and sends them to P_3 and allocates agents for them who can perform that task.
t_2 : It will be fired for those tasks in P_1 place which are dependent on one or more tasks of the system and sends them to P_4.
t_3:It will be fired for only those tasks with allocated agents who get their required resources and sends them to P_5.
t_4:It will be fired for only those tasks with allocated agents who do not get their required resources and sends them to P_6.
t_{10} :It will be fired if token i.e the task in P_9 finishes successfully & send the agent which finishes the task successfully to P_{15}.
t_{11} :It will be fired if token i.e the task in P_9 with allocated agents abort due to crash of agent and send that incomplete task to P_{11}.
t_{12} : It will be fired if task in P_9 finishes successfully and sends back all dependent tasks of that task with allocated agents to P_3 if they are not further dependent on any other task.
t_{13} :It will be fired if there are tokens both in P_9, P_{10} and used to pass control information i.e timer and state information.
t_{15} : It will be fired to select one of the dependent task with allocated peer agent and send the agent along with the incomplete faulty task to P_9 to finish that, and send other dependent tasks of that agent to P_{13}.
t_{16} : It will be fired if faulty agent of P_{11} finishes successfully and this transition send back a copy of employed agent from P_{15} along with its tasks at P_{13} to P_3.
t_{19} : It will be fired if single task in P_{14} finishes successfully and sends that to P_{15}.
t_{20} It will be fired if faulty task of P_{17} finishes successfully and a copy of employed peer agent at P_{15} along with the rest of dependent tasks is sent back to P_3.
t_{21}:It will be fired if single agent in P_{14} crashes and sends the incomplete task to P_{17}.
t_{22} :It will be fired for only to transfer control information between agents of P_{14} and P_{16}.
t_{25} : It will be fired to employ one peer agent to do the faulty task of P_{17} and send rest of the tasks of that agent to P_{20}.
t_{28} It will be fired if there are multiple agents with allocated task in P_5 and send a timer tm to each of these agents and along with this timer send them to P_{21}.
t_{29} : this transition will be fired if any agent of P_{21} crashes during execution and send that incomplete task to P_{22}.
t_{30} : this transition will be fired only to pass timer information between agents of P_{21}.
t_{33} : this transition will be fired to employ one peer agent from P_{24} to finish faulty task of P_{22} and send that agent to P_{21} and send rest tasks of that agent to P_{25}.
t_{34} : It will be fired if any agent of P_{21} finish successfully and send
that agent to P_{15}.
t_{35} : It will be fired after finishing successfully the faulty task of P_{22} and sent that employed agent along with its rest task to P_3.
t_{38} : It will be fired to send control information between agents of P_{26} and P_{14}.
t_{39} : It will be fired if the agent of P_{14} crashes and any one agent from agent pool P_2 is employed by any agent of P_{26} and send that agent to P_{14} to complete the incomplete task.
t_{40} : It will be fired if there is a single agent ready to run with no concurrent agent in waiting or running state and no dependent agent, and then send 5 copies of that single agent to P_{27}.
t_{41} : It will be fired to exchange control information between single
agent of P_{14} and agents of P_{28}.
t_{44} : It will be fired if the faulty task of P_{17} finishes successfully and if the rest tasks of employed peer agent in P_{20} in are still dependent then send those task to P_4 place and send that peer agent to agent pool P_2.

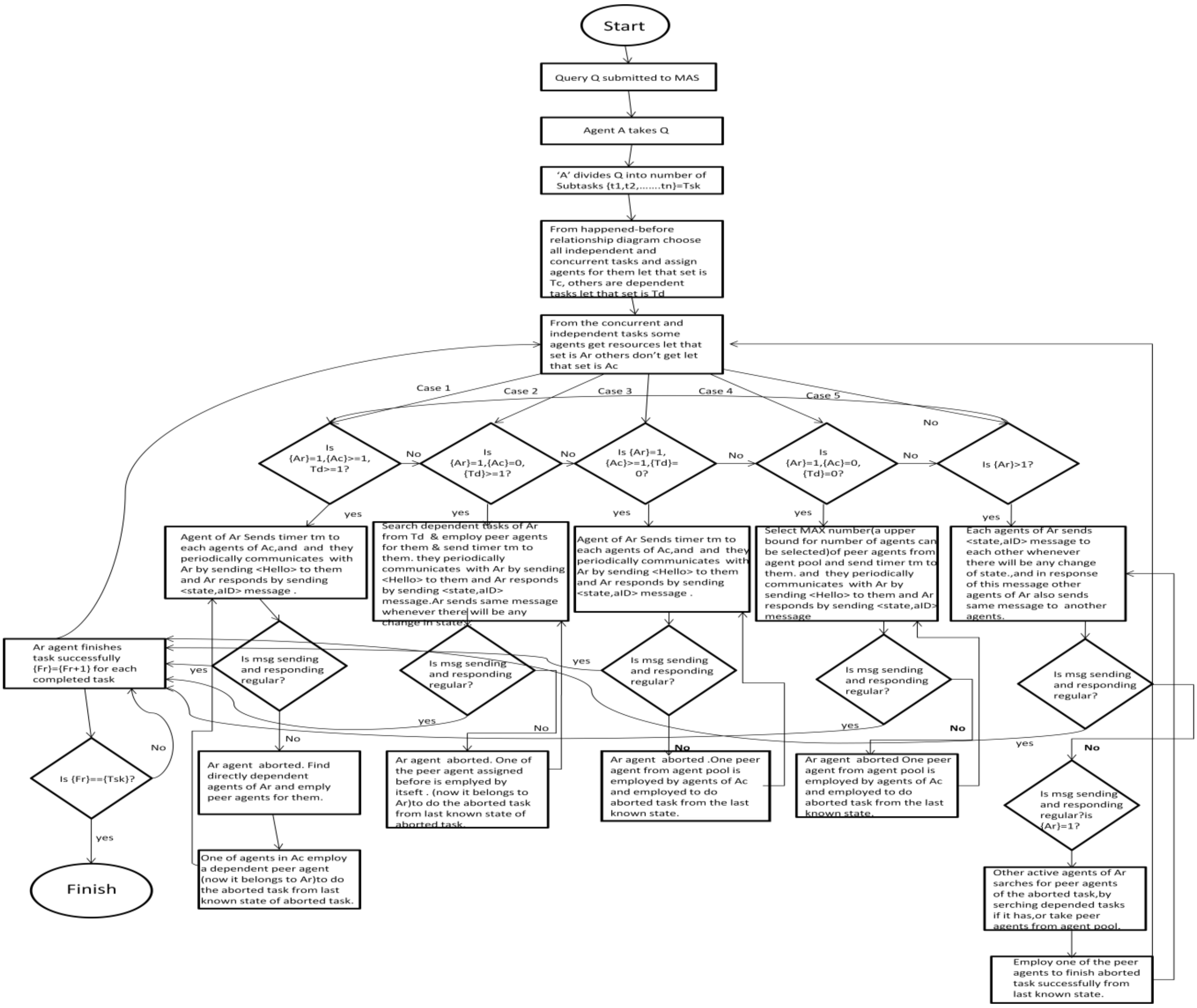

Figure 1: Flow chart of our proposed protocol

t_{45} : It will be fired if the faulty task of P_{17} finishes successfully and if the dependent tasks of that finished task in P_{19} are still dependent on any other task then send those tasks to P_4 and send allocated agents for those tasks to agent pool p_2.

t_{46} : It will be fired if the faulty task of P_{11} finishes successfully and if the rest tasks of employed peer agent in P_{13} are still dependent then send those task to P_4 place and send that peer agent to agent pool P_2.

t_{47} : It will be fired if the faulty task of P_{11} finishes successfully and if the dependent tasks of that finished task in P_{10} are still dependent on any other task then send those tasks to P_4 and send allocated agents for those tasks to agent pool p_2.

t_{49} : It will fire if P_{21} contains single token and send that token to P_{14}.

t_{50} :It will be fired if the faulty task of P_{22} finishes successfully and if the rest tasks of employed peer agent in P_{25} in are still dependent then send those task to P_4 place and send that peer agent to agent pool P_2.

t_{51} It will be fired if the faulty task of P_{22} finishes successfully and if the dependent tasks of that finished task in P_{25} are still dependent on any other task then send those tasks to P_4 and send allocated agents for those tasks to agent pool p_2.

t_{52} : It will be fired if any faulty agent in P_{22} has no dependent agents then one agent from agent pool who can perform the faulty task is chosen by other agents executing in P_{21} and send that agent to P_{21} to finish the incomplete task. Other transitions will be fired according to the basic Petri-net concept.

5.ANALYSIS

In this section, we will prove that the protocol we have defined is fault tolerant for every scenario in MAS. For each of the 5 cases shown below, the final marking of the system after finishing a task is reachable from a faulty state where agent performing that task aborts during execution.

Figure 2: Petri net representation of our proposed protocol

Let us take an example to prove our assumptions. Let a user submit a query Q, which is divided into 5 sub-tasks (tsk1, tsk2, tsk3, tsk4, tsk5) by the interface agent. The happened before relationship between those divided tasks can be represented by a dependency graph given below:

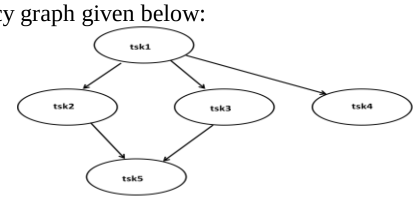

Figure 3: task dependency graph of subtasks of query Q

Initially there are 5 subtasks, which are kept in the starting place of petri-net (place P1). Let us assume the number of agents in agent pool (place P2) is 20. Initially there are no tokens in other places of this petri-net.The incidence matrix of the Petri net represented in figure 2 is denoted by I.

*Lemma 1. Single agent executing a task in the system with no concurrent tasks but have dependant tasks, can support fault tolerance***.**

Proof: In our example this situation will occur when tsk1 is ready to be executed. It has three directly dependant tasks tsk2, tsk3 and tsk4 and one indirectly dependant task tsk5 but no concurrent tasks. From our petri-net diagram we can see Initial marking before starting execution of any tasks is M_i = [5 20 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'. If agent performing tsk1 aborts during execution the fault state marking is M_j = [0 16 0 1 0 0 0 0 0 3 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'. We can reach from M_i to M_j through following sequence to transitions, σ_1 =[1 4 1 0 3 1 0 1 3 0 1 0 2 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'. If our MAS is fault tolerant its leads to generate a final state of system after completing tsk1 successfully even after agent performing tsk1 aborts, i.e final state after successful completion of tsk1 is M_k = [0 16 3 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0] and we get a sequence of transitions to reach from M_j to M_k which is, σ_2 = [0 0 0 0 0 0 0 0 0 1 0 2 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'.

Putting these in the equation (1),

[5 20 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]' + I. σ_1 = [0 16 0 1 0 0 0 0 0 3 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'

[0 16 0 1 0 0 0 0 0 3 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]' + I. σ_2 =[0 16 3 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'

So we can say in this scenario MAS we have designed is fault tolerant, i.e final state after completing that single task is reachable from fault state.

Lemma 2. Single agent executing a task in the system with concurrent tasks and dependant tasks can support fault tolerance.

Proof: In Lemma 1 we have proved tsk1 reaches to final state successfully even after agent which is performing tsk1 crashes. Now three dependant tasks tsk2, tsk3, tsk4 can run concurrently. Now situation mentioned in lemma 2 will occur when tsk2 gets resources and goes for execution and tsk3, tsk4 are not ready as they don't get resources and tsk5 is in dependant task place. In this scenario the initial marking (last marking of lemma 1) is M_i = [0 16 3 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'. If tsk2 aborts during execution the fault state marking is M_j = [0 16 0 1 0 0 0 0 0 0 0 0 0 0 1 2 1 0 0 0 0 0 0 0 0 0 0 0 0]'. We can reach from M_i to M_j through following sequence to transitions σ_1 = [0 0 1 2 0 1 0 0 0 0 0 0 0 0 0 0 0 1 2 0 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'. If our MAS is fault tolerant it leads to a final state of system after completing tsk2 successfully even after agent performing tsk2 aborts, i.e final state after successful completion of tsk2 is M_k=[0 16 2 1 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]' and we get a sequence of transitions to reach from M_j to M_k which is, σ_2 =[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]'

Putting in the equation (1),

[0 16 3 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'+ I. σ_1 =[0 16 0 1 0 0 0 0 0 0 0 0 0 0 1 2 1 0 0 0 0 0 0 0 0 0 0 0 0]'

[0 16 0 1 0 0 0 0 0 0 0 0 0 0 1 2 1 0 0 0 0 0 0 0 0 0 0 0 0]' + I. σ_2 =[0 16 2 1 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'

So we can say in this scenario MAS we have designed is fault tolerant, i.e final state after completing that single task is reachable from fault state .

Lemma 3. Multiple agents executing more than one task in the system can support fault tolerance.

Proof: In Lemma 1 and Lemma 2 we have p**r**oved tsk1, tsk2 reaches to final state successfully even after agents which are performing tsk1and tsk2 abort. Now tsk3 and tsk4 can run concurrently. Now situation mentioned in Lemma 3 will occur if both tsk3 and tsk4 get resources and start execution.tsk5 is in dependant task place. In this scenario the initial marking (last marking of lemma 2) is M_i=[0 16 2 1 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'. If tsk3 aborts during execution the fault state marking is, M_j=[0 16 0 1 0 0 0 1 0 0 0 0 0 0 2 0 0 0 0 0 0 1 1 0 0 0 0 0 0]'. We can reach from M_i to M_j through following sequence to transitions, σ_1 =[0 0 2 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'. If our MAS is fault tolerant it leads to generate a final state of system after completing tsk3 successfully even after agent performing tsk3 aborts, i.e final state after successful completion of tsk3 is M_k=[0 15 2 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]' and we get a sequence of transitions to reach from M_j to M_k which is, σ_2 =[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0]'

Putting in the equation (1),

[0 16 2 1 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]' + I. σ_1 = [0 16 0 1 0 0 0 1 0 0 0 0 0 0 2 0 0 0 0 0 0 1 1 0 0 0 0 0 0]'

[0 16 0 1 0 0 0 1 0 0 0 0 0 0 2 0 0 0 0 0 0 1 1 0 0 0 0 0 0]' + I. σ_2 =[0 15 2 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'

So we can say in this scenario MAS we have designed is fault tolerant, i.e if any one or more agents among multiple running agents abort, then also those tasks can be finished successfully and reaches to a final state.

Lemma 4. Single agent executing tasks in the system with concurrent tasks but no dependant tasks can support fault tolerance.

Proof: In Lemma 1, Lemma 2 and Lemma 3 we have proved tsk1, tsk2, tsk3 reach to final state successfully even after agents which are performing tsk1, tsk2, tsk3 abort. Now tsk4 and tsk5 can run concurrently. Now situation mentioned in Lemma 4 will occur if tsk4 gets resources and starts execution and tsk5 does not get resources. There is no task in dependent place. In this scenario the initial marking (last marking of lemma 3) is, M_i=[0 15 2 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'. If tsk4 aborts during execution the fault state marking is, M_j=[0 15 0 0 0 0 0 0 0 0 0 0 0 0 3 0 1 0 0 0 0 0 0 0 0 1 0 0]'. We can reach from M_i to M_j through following sequence to transitions , σ_1 =[0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'. If our MAS is fault tolerant its leads to generate a final state of system after completing tsk4 successfully even after agent performing tsk4 aborts, i.e final state after successful completion of tsk4 is M_k=[0 14 1 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0]' and we get a sequence of transitions to reach from M_j to M_k which is, σ_2 =[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0]'

Putting in the equation (1),

[0 15 2 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'+ I. σ_1 =[0 15 0 0 0 0 0 0 0 0 0 0 0 0 3 0 1 0 0 0 0 0 0 0 0 1 0 0]'

[0 15 0 0 0 0 0 0 0 0 0 0 0 0 3 0 1 0 0 0 0 0 0 0 0 1 0 0]'+ I. σ_2 =[0 14 1 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'

So we can say in this scenario MAS we have designed is fault tolerant, i.e final state after completing that single task is reachable from its fault state.

Lemma 5. Single agent executing in the system with no concurrent tasks and no dependant tasks can support fault tolerance.

Proof: In Lemma 1, Lemma 2, Lemma 3 and Lemma 4 we have proved tsk1, tsk2, tsk3, tsk4 reach to final state successfully even after agents which are performing tsk1, tsk2, tsk3, tsk4 abort. Now only tsk5 is in concurrent state. Now situation mentioned in Lemma 5 will occur if tsk5 gets resources and becomes ready to be executed. There is no task in concurrent and dependent place. In this scenario the initial marking (last marking of lemma 4) is, M_i=[0 14 1 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'. If tsk5 aborts during execution the fault state marking is, M_j=[0 9 0 0 0 0 0 0 0 0 0 0 0 0 4 0 1 0 0 0 0 0 0 0 0 0 5 5]'. We can reach from M_i to M_j through following sequence to transitions , σ_1 =[0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 1 0 0 0 0 0 0 0 0 0 0 0]' if our MAS is fault tolerant its leads to generate a final state of system after completing tsk5 successfully even after agent performing tsk5 aborts, i.e final state after successful completion of tsk5 is, M_k=[0 9 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 0 5 4]' and we get a sequence of transitions to reach from M_j to M_k which is, σ_2 =[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]'

Putting in the equation (1),

[0 14 1 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'+ I. σ_1 =[0 9 0 0 0 0 0 0 0 0 0 0 0 0 4 0 1 0 0 0 0 0 0 0 0 0 5 5]'

[0 9 0 0 0 0 0 0 0 0 0 0 0 0 4 0 1 0 0 0 0 0 0 0 0 0 5 5]'+ I. σ_2 =[0 9 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 0 5 4]'

So we can say in this scenario MAS we have designed is fault tolerant, i.e final state after completing that single task is reachable from its fault state .It is the last and final state of given query. So, from above mentioned five Lemmas we prove that even after a number of failures in MAS all tasks can be completed successfully and final state after executing a query can be reached successfully.

6. CONCLUSION

In this paper, we have proposed fault detection and handling mechanism and have verified its correctness with the help of color petri nets. We have shown that MAS achieves the goal state even if some faults occur in the system. This protocol assumes that whenever an agent crashes, its peer agent will be present in healthy state in the agent pool. The future prospect of this work is to consider cases when no peer exists in the agent pool. Another prospect of this work is to implement the protocol and determine the degree of fault tolerance i.e. to what extent can the system provide fault tolerance.

7. REFERENCES

[1] Weiss, G. (Ed). 1999. Multiagent systems: a modern approach to distributed artificial intelligence. MIT Press.

[2] Wooldridge, M. J. 2001. Introduction to Multiagent Systems. John Wiley & Sons.

[3] Yen , H. 2006. Introduction to Petri Net Theory, In Recent Advances in Formal Languages and Applications. volume 25/2006 of Studies in Computational Intelligence,pages 343–373.

[4] Suzuki, T., Shatz, M., and Murata, T. 1990. A Protocol Modeling and Verification Approach Based on Specification Language and Petri Nets . IEEE Transactions on Software Engineering, vol. 16. no. 5.May 1990, 523-536.

[5] Bai, Q., Zhang, M. and Zhang, H. 2005. A Coloured Petri Net Based Strategy for Multi-agent Scheduling. In the proceedings of the Rational, Robust, and Secure Negotiation Mechanisms in Multi-Agent Systems.

[6] Jindian, S., Heqing, G., and Shanshan, Y. 2008. A Coloured Petri Net Model for Composite Behaviors in Multi-Agent System. In Proceedings of IEEE Conference on Cybernetics and Intelligent Systems, 677 – 680.

[7] Tanenbaum, A.S. 1995. Distributed Operating Systems. Pearson Education.

[8] Fedoruk, A., and Deters, R. 2002. Improving fault-tolerance by replicating agents. In Proceedings of Autonomous agent and multi agent systems, AAMAS-02, Bologna, Italy, 737-744.

[9] Guerraoui, R., and Schiper, A. 1997. Software-based Replication for Fault Tolerance. IEEE Computer, vol. 30, no. 4, 68-74.

[10] Almeida, A., Aknine,S., Briot,J.P., and Malenfant, J. 2006. A Predictive Method for Providing Fault Tolerance in Multi-Agent Systems. In proceedings of IEEE/WIC/ACM International conference on Intelligent agent technology (IAT 2006),Hong kong.

[11] Hugg, S. 1997. A Sentinel Approach to Fault Handling in Multi-Agent Systems. In the proceedings of the 2nd Austrdian Workshop on Distributed AI, Caims, Australia.

[12] Kumar,S., Cohen.,P.R., and Levesque,H.J. 2000. The adaptive agent architecture: achieving fault tolerance using persistent broker teams. In Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS 2000), Boston, MA, USA.

[13] Mirian, M.S., Abmadabadi,M.N., and Navabi,Z. 2002. A decision based approach for fault handling in multi agent system. In the proceedings of the 9th international Conference on Neural Information Processing (ICONIP'OZ) , Vol. *4*.