A Consensus Algorithm for Synchronous Distributed Systems using Mobile Agent

Jiannong Cao¹, Xianbing Wang^{1,2}, Siu Lo¹, Sajal K. Das³

¹Internet Computing and E-Commerce Lab,

Department of Computing, Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong

²Computer Center, School of Computer, Wuhan University, Wuhan, Hubei, China

³Center for Research in Wireless Mobility and Networking(CrewMan),

Dept. of Computer Science and Engineering, University of Texas at Arlington,
Arlington, Texas 76019, USA

Abstract

In this paper, we present a consensus algorithm for synchronous distributed systems using cooperating mobile agents. The algorithm is designed within a framework for mobile agent enabled distributed server groups (MADSG), where cooperating mobile agents are used to achieve coordination among the servers. Being autonomous and cooperative, cooperating mobile agents exchange information among themselves to achieve consensus. The correctness proof of the proposed algorithm is presented. Performance evaluation of the algorithm has been carried out by both theoretical analysis and simulation.

1. Introduction

A mobile agent is a computer program that can autonomously migrate among network sites, i.e., it can execute at a host for a while, halt execution, dispatche itself (together with its data and execution state) to another host, and resumes execution there - all under its own control. Characteristics of mobile agents include mobility, autonomy, asynchronous execution, support for mobile computing, etc. Recent years have seen an explosion of interest in mobile agent technology and its applications [14, 20].

Cooperating mobile agents [3, 4, 21] are a collection of mobile agents, which come together for the purpose of exchanging information or in order to engage in cooperative task-oriented behaviors. In our previous work, we have proposed a framework, called MADSG, for mobile agent enabled distributed server groups. Using cooperating mobile agents, MADSG provides a novel system architecture by having mobile agents carrying out coordination tasks for the cooperating servers. In traditional distributed client/server systems using message passing based approaches, the coordination code has to be integrated into the server service model itself. This mixes the functionality of providing services with the service

independent operation for maintaining data consistency and ensuring performance. Also, whenever a new coordination protocol is used or a new feature is introduced, the server code must be re-implemented. Using mobile agents allows us to provide clear and useful abstractions through the separation of different concerns. The server site functionality can be separated from the operations of maintaining the logical relationship between group members and providing the desired level of performance, which are realized by a collection of autonomous, cooperating mobile agents. This helps reduce the complexity of and increase the flexibility in implementing the servers. Because mobile agent technology is innately suitable for disconnected and mobile computing [14], the framework could also support cooperation among a group of disconnected and mobile users.

MADSG provides a general, flexible framework in which distributed control functions (such as checkpointing, load sharing, distributed mutual exclusion, replication, etc.) for a wide range of distributed applications can be achieved by using cooperating mobile agents [2,3,4,5]. Cooperating mobile agents encapsulate policies and algorithms for their interaction and coordination in order to implement various distributed control functions. In the paper, we present a mobile agent enabled algorithm for solving the consensus problem in a synchronous distributed system.

Consensus is a fundamental problem in distributed systems. It states that there are a set of n processes, $\{p_1, p_2, \dots, p_n\}$, each process p_i initially proposes a value v_i , and all non-faulty processes have to decide on one common value v which is equal to one of the proposed values. Obviously, consensus only becomes a problem when some processes in the distributed system can fail. A process *fails* if it behaves abnormally, e.g., by crashing. A process is *correct* if it never fails. Consensus has been extensively studied over last two decades both in synchronous and asynchronous distributed systems [1,6,7,8,9,10,11,12,13,15,18,19,22]. In a synchronous system, message delays and relative processes speeds are

bounded and these bounds are known. In contrast, none of these bounds exist in an asynchronous system. In [13], Fischer, etc. proved that consensus can't be solved deterministically in an asynchronous system that is subject to even a single crash failure. The impossibility result for consensus stems from the inherent difficulty of determining whether a process has actually crashed or is only very slow. The research in asynchronous model focused on the use of randomization techniques [9], partial synchrony [12], and unreliable failure detectors [1,6].

In this paper, we consider the consensus problem in synchronous systems with crash failures. Assume there are n hosts in the distributed system. In our algorithm, some of the n hosts will be designated as *Coordinating Hosts (CHs)*. Each host delegates mobile agents to CHs for exchanging information. The host creating a mobile agent is called the *home host* of that agent. Once being dispatched, a mobile agent executes autonomously and coordinates with the mobile agents from other hosts to help achieve consensus among the hosts. The design of our mobile agent-enabled algorithm uses the *Pigeon Hole Principle*. Initially, $t+1$ hosts are randomly chosen to server as the, where t ($t < n$) is the maximum number of hosts that can crash and. Therefore, there is at least one *CH*, which never crash.

The algorithm is round-based. In each round, a host dispatches a group of mobile agents, one for each live *CH*. On arriving at a *CH*, all mobile agents exchange information with each other, and then return to their home hosts. When all agents dispatched from a host return and no new information is obtained during the current round, the host decides based on the current value set and then terminates. Because there is at least one correct *CH* and correct *CHs* never destroy mobile agents, the information brought back by the mobile agents must be complete and reliable.

Our proposed algorithm achieves uniform consensus with the upper bound $\lceil (t+1)/2 \rceil$. That is, in the presence of up to t crash failures, uniform consensus can be solved by our algorithm within $\lceil (t+1)/2 \rceil$ rounds. Note that the meaning of the notion *round* used in this paper is different from that in traditional message-passing consensus algorithms [6,7,8,11,18,19]. In those algorithms, in round r , each process executes sequentially the following steps: 1) sends a r th-round message to the other processes, 2) waits for a r th-round message from the other processes, and 3) executes local computations. The time cost of a round in our algorithm is nearly the double of the time cost of a round in traditional synchronous consensus algorithms. In our algorithm, early stop [11] can be achieved, uniform consensus can be solved within $\lceil (f+1)/2 \rceil + 1$ rounds where f ($f < t$) is the number of crashes that actually occur. Unlike traditional early stopping consensus algorithms in synchronous systems, such as *F_OptFloodSet* [7] and *EDAC* [8], which every process

need to broadcast a *decide* message after it has made decision, our algorithm need not do this.

The rest of the paper is organized as follows. Section 2 describes the MADSG framework and the system model. In section 3, we describe the mobile agent enabled consensus algorithm and prove the correctness of the proposed algorithm. Section 4 presents the performance analysis. Section 5 describes the simulation evaluation of the algorithm. Finally, section 6 concludes this paper.

2. System model

We consider a group of distributed servers, e.g., web servers and database servers, which cooperate to provide certain services, such as information processing, trading, network management, and computation. Figure 1 illustrates the MADSG framework, where mobile agent is used as an aid to the servers for achieving their coordination. Distributed control functions, which maintain logic relationship between group members and provide the desired level of performance, such as checkpointing, load sharing, distributed mutual exclusion, consensus, etc. are realized by a collection of autonomously cooperating mobile agents. Since the mobile agent carries the program it executes in its entirety as it propagates through the network, utilizing the information provided by the local servers, a flexible and adaptive coordination scheme could be developed.

Figure 1. The MADSG framework

Each server node hosts a mobile agent server which is responsible of creating, executing, and destroying mobile agents. Mobile agents have their own identity and behaviour and are capable of navigating through the underlying network, performing various tasks at the sites they visit, and communicating with other mobile agents. Mobile agents are sent by or on behalf of the server

processes for coordinating their activities. Individual server has a well defined interface with primitives that can be invoked by mobile agents to access the information maintained at the server, and to perform some local operations.

There are totally n servers $\{h_1, \dots, h_n\}$ (hereafter, referred to as hosts). Every host has a unique identity over the network. We assume a synchronous distributed system model [19].

There are several failure models in solving the consensus problem [16]: (1) Crash, in which processes stop executing and fail to send messages, (2) Omission, in which communication channels may omit messages, and (3) Arbitrary (Byzantine), in which processes may act maliciously. The crash failure can be classified into *crash/stop* model, if the crashed process cannot recover, and *crash/recovery* model, if the crash process can recover during the execution of the consensus protocol [1]. We consider the crash/stop failure model in this paper. A host failed means it crashed and can no longer dispatch mobile agents. Up to t ($t < n$) out of the n hosts can crash in the system.

Hosts can create and dispatch mobile agents to any other hosts through the reliable underlying communication network. The network is reliable in the sense that it does not create new mobile agents and does not modify the propagating agents. In addition, it does not lose agents. Therefore, if a mobile agent is lost, it must be crashed in a host while it fails.

For simplicity, we assume that, when dispatching mobile agents, the home host does not try to detect whether the destination host is crashed. Instead, it just dispatches an agent towards the receiver host without waiting for any acknowledgement. If the destination host is already crashed, the agent is assumed to get lost.

In our setting, when the hosts in the group want to achieve consensus, each of the n hosts, h_i , will propose a value v_i and then dispatch its coordinating mobile agents. All correct hosts will decide on some value v , in relation to the set of proposed values $V = \{v_i \mid i = 1, \dots, n\}$. More precisely, the Consensus problem is defined by the three following properties:

- **Termination:** Every correct host eventually decides some value.
- **Validity:** If a host decides v , then v was proposed by some host.
- **Agreement:** No two correct hosts decide differently.

The agreement property applies to only correct hosts. Thus, it is possible that a host decides on a distinct value just before crashing. Uniform Consensus prevents such a possibility. It has the same Termination and Validity properties but the following agreement property:

- **Uniform Agreement:** No two hosts (correct or not) decide differently.

In this paper, we consider the uniform consensus problem.

3. The Mobile agent enabled consensus algorithm

3.1 Informal Description

$t+1$ hosts are randomly chosen as *Coordinating Hosts (CHs)*. For simplicity, we choose the first $t+1$ hosts, $h_1, \dots, h_{t+1}$, as *CHs*. Therefore, there is at least one *CH* which never crashes. We call such a *CH* a *correct CH*.

Our algorithm is round-based. In each round, mobile agents for the hosts perform the following two steps:

- Dispatch themselves to their designated live *CHs*, exchange information with agents from other hosts.
- Return to their home hosts to determine whether the decision can be made by applying the consensus function to the current state and the new information brought back by the mobile agents.

Mobile agents from each host execute the protocol in synchronous rounds. The algorithm terminates when all correct hosts have made decisions.

On each live *CH*, the mobile agents exchange information with each other, and then return home. On returning home, a stationary agent, called the *master agent*, performs calculations on the value set and a flag of every mobile agent to obtain the information that indicate whether the algorithm can terminate or not. If execution of the algorithm is not yet to complete, another round starts. Otherwise, the algorithm terminates, making decision based on the current value set, and all mobile agents stop their executions. The decision on termination is made as follows. Every mobile agent carries a flag, *Altered_Flag*, which indicates whether some new values are added into its value set during last trip. The flags of all agents created by a host will be combined after they return home. If there exist at least one agent with its flag set to be true, the next round is started for this host.

3.2 The Proposed Algorithm

Algorithm 1 is executed by a mobile agent, $Agent_{i,j}$, on behalf of host h_i traveling to CH_j . $V_{i,j}$ denotes the value set carried by $Agent_{i,j}$ and V_h denotes the value set derived from all the value sets carried by the mobile agents at a host h . T_s denotes the maximum time required by an agent to migrate between any two hosts. $f_{i,j}$ denotes the *Altered_Flag* of Agent $Agent_{i,j}$ (line 2). The algorithm terminates as soon as the agent is destroyed by its home host (see explanation for **Algorithm 2**).

In each iteration of the loop, the mobile agent $Agent_{i,j}$ waits for the signal for its migration (line 5). After received signal from its home host, it then copy the value set maintained by the master agent and migrates to CH_j (line 6-7). After arrival at CH_j , the agent gives its value set to CH_j (line 8). Then it waits a signal from CH_j . When received signal, every agent, which is dispatched to CH_j , must have arrived at CH_j (line 9), and V_{CH_j} will contain complete values of all arriving agents. $Agent_{i,j}$

then compares V_{CH_j} and the value set V_{ij} , and sets the flag f_{ij} to indicate whether they are different (line 10). Finally, the agent copies the value set of CH_j (line 11) and returns to its home host (line 12).

Algorithm 1 executed by a mobile agent

```

1. Initialization
2. define  $f_{ij}$  as Altered_Flag of  $A_{ij}$ 
3. Algorithm
4. while (alive)
5.   wait_signal() //wait signal from its creating host
6.    $V_{ij} := V_i$ ;
7.   migrate_to( $CH_j$ )
8.    $V_{CH_j} := V_{CH_j} \cup V_{ij}$ 
9.   wait_signal() //wait signal from the CH
10.   $f_{ij} := (V_{CH_j} \neq V_{ij}) ? \text{true} : \text{false}$ 
11.   $V_{ij} := V_{CH_j}$ 
12.  migrate_to( $h_i$ ) // return home
13. end while

```

Algorithm 2 is executed by the master agent at each host h_i . The master agent first initializes the variables at line 2 to 4. There are total $t+1$ CHs, namely CH_0, CH_1 , until CH_t . V_i initially contains its proposed value, v_i . The master agent maintains an agent set, A_i , for holding mobile agents created by itself. Initially it creates and initializes $t+1$ mobile agents, namely, $a_{i,0}, a_{i,1}, \dots, a_{i,t}$. All these mobile agents are added to the set A_i .

Then the master agent sends a signal to the agents in A_i (line 7) for migration. For a master agent on a Coordinating Host, if it has not made the decision, it wait for T_s for all agents migrating to it to arrive (line 9), collect the information, and then signal the agents to bring the updated information home (line 10). It will then wait for another T_s for all its agents to come back (line 11). Master agents on other hosts just wait for $2T_s$ for its agents to return home (line 12). An agent migrated to a CH which has already made its decision will not come back.

Upon timeout, if no agent returns (line 15), which means all live CHs have made decision and terminated, the master agent will set f_i to false to exit the loop and make decision. Otherwise, it calculates the intersection set of value sets of all agents in the set A_i , and store the result set into its own value set V_i (line 17). It applies the “OR” operation on Altered_Flag variables of all agents in the set A_i , and store the result into the local variable f_i (line 18).

If the f_i is true, implying that some agents in A_i reported that the values collected from their CHs are not consistent, the master agent will go back to line 7 to start a new round. If not, the loop exits and all mobile agents will be destroyed (line 21), and the decision is made according to the current value set of host h_i (line 22).

Algorithm 2 executed by a master agent

```

1. Initialization
2.  $V_i := \{v_i\}$ 
3.  $A_i := \emptyset$ 
4. for each  $CH_j$ , create_agent( $a_{ij}$ ), init( $a_{ij}$ ),  $A_i := A_i \cup \{a_{ij}\}$ 
5. Loop
6.   repeat
7.     for each  $a_{ij} \in A_i$ , signal( $a_{ij}$ )
8.     if itself is a Coordinating Host then
9.       wait_timeout( $T_s$ ) // all agents migrate to it can arrive
10.      signal all arrived agents
11.      wait_timeout( $T_s$ ) // its all agents can return
12.      else wait_timeout( $2T_s$ ) // its all agents can return
13.      end if
14.       $A_i := \{ \text{all agents that came back} \}$ 
15.      if  $A_i = \emptyset$  then  $f_i := \text{false}$ 
16.      else
17.         $V_i := \bigcap_{a_{i,j} \in A_i} V_{i,j}$ 
18.         $f_i := \text{OR}_{a_{i,j} \in A_i} f_{i,j}$ 
19.      end if
20.      until  $f_i = \text{false}$ 
21.       $A_i := \emptyset$  // delete all agents
22.       $d_i := \min(V_i)$ 

```

3.3 Correctness Proof

A consensus algorithm is correct if it preserves the three properties of consensus. The properties are validity, termination and uniform agreement. In this section we prove that our algorithm does preserve the properties.

Lemma 1. The value set V_i is always nonempty.

Proof. According to the algorithm, initially, $V_i = \{v_i\}$. Because when a agent dispatched from h_i arrives on a CH, it will combine V_i with the value set of CH, V_{CH} . The agent comes back with the new V_{CH} . Therefore, we always have $V_i \subseteq V_{CH}$ for each returning agent. So, after performing intersection, $\{v_i\} \subseteq V_i$ will always be true. Thus the Lemma must be true. $\square$

Proof for Validity

The proof is straightforward. Because a host makes decision based on the values obtained from the value sets brought back by its mobile agents using only intersection and union operations, by Lemma 1, the set of values will always be nonempty and contain only the original proposed values.

Proof for Termination

The condition to terminate **Algorithm 2** at a host is that f_i of the master agent becomes *false* at a round, which means, in such a round, no agent modifies its value set. This condition can be true if no host crashes in the previous round. Obviously, this can eventually happen because the number of hosts that can fail is up to t . Therefore, in the worst case, the algorithm can

terminate in one of the first $(t + 2)$ rounds. When **Algorithm 2** terminates, the executing master agent will kill all its mobile agents thus terminate the executions of **Algorithm 1**.

Lemma 2. At the end of a round, if a host h_i has some of its mobile agents returned and finds that f_i is false, it will make decision. In this case, all other live hosts maintain the same value sets as V_i .

Proof. Assume h_i makes decision at round r . Because no change occurs to the value set of any agent created by h_i , we have $V_i^r = V_i^{r-1}$. For any host h_j that has not crashed after round r , V_i^{r-1} must belong to the value set of every returning agent of h_j , so $V_i^{r-1} \subseteq V_j^r$. Because there are $t+1$ CHs and at most t hosts can crash, there exists at least one CH, from which both h_i and h_j have an agent return. After all agents of h_j return, h_j will perform an intersection on the value sets those agents carried, resulting $V_j^r = V_i^{r-1} = V_i^r$. Thus the Lemma must be true. $\square$

Proof for Uniform Agreement.

There are two cases for a host to make decision and terminate: (1) upon in return of its agents, it finds that no changes has occurred to the value set of these agents; (2) no agent returns which means all live CHs have made decision and terminated. In both cases, by **Lemma 2**, when the first host makes decision (of course, it must be in the first case), all other live hosts maintain the same value set. According to the algorithm, their value sets will never be changed. Thus they will make the same decision.

4. Performance Analysis

In this section we analyze the performance of the proposed algorithm. The performance metrics used are the time required for achieving consensus and the number of mobile agent migrations.

4.1 Upper Bound on the Number of Rounds

Lemma 3. If only one host crashes in a round, all live hosts will maintain the same value set at the end of the round.

Proof. Without the loss of generality, assume h_i crashes in round r . According to the algorithm, a round can be divided into two steps. In the first step, every live host dispatches agents to all live CHs to exchange information. In the second step, these agents return home to determine if the decision can be made. Now we can classify the crash status into four cases:

1. h_i is not a CH and crashes in the first step of a round, and some of its agents may not be dispatched. In this case there are no CH crashing, every live hosts will have an agent returning from each CH which maintains the same value set. Thus at the end of the round, all live hosts will maintain the same value set.

2. h_i is not a CH and crashes in the second step of a round, in this case it has dispatched to each live CH an agent. As the same reason as in the case 1, all live hosts will maintain the same value set.

3. h_i is a CH and crashes in the first step of a round. All live hosts can still successfully receive the same number of agents from other live CHs and, at the end of the round, maintain the same value set.

4. h_i is a CH and crashes in the second step of a round. Because in the first step, all other live CHs will receive an agent from each live host including h_i , these CHs will maintain the same value set. Though h_i crashes in the second step, all live hosts can maintain the same value set.

In all cases, live hosts can maintain the same value set. Thus the Lemma must be true. $\square$

Theorem 1. The upper bound on the number of rounds for the proposed algorithm is $\lceil (t+1)/2 \rceil + 1$.

Proof. Assuming the contrary, an execution of the algorithm needs $\lceil (t+1)/2 \rceil + 2$ rounds. According to **Lemma 3**, in each of the first $\lceil (t+1)/2 \rceil$ rounds, at least two hosts crashed. Otherwise, all live hosts will maintain the same value set by the end of the $\lceil (t+1)/2 \rceil$ th round, and they will make decision by the end of the $(\lceil (t+1)/2 \rceil + 1)$ th round. Because at least two hosts crashed in each round, the total number of hosts crashed in the first $\lceil (t+1)/2 \rceil$ rounds will be at least $(\lceil (t+1)/2 \rceil) \times 2 > t$. This cannot be true. Thus the theorem must be true. $\square$

Corollary 1. Uniform consensus can be achieved by our algorithm within $\lceil (t+1)/2 \rceil + 1$ rounds according to the number of failures f actually occur, where $f \leq t$.

The proof is the same as Theorem 1.

4.2 An Improvement on the Upper Bound

According to **Theorem 1**, if an execution of the algorithm reaches the $\lceil (t+1)/2 \rceil$ th round, all live hosts will maintain the same value set. Therefore, they can make decision either before or at the end of the $(\lceil (t+1)/2 \rceil + 1)$ th round. With this observation, we can improve the proposed algorithm to let all live hosts make decision and terminate at the end of the $\lceil (t+1)/2 \rceil$ th round if they can reach this round.

To achieve the improvement, **Algorithm 1** does not need any change and we just need to modify **Algorithm 2** as follows. A variable r (line 5 in the improved algorithm) is added to represent the round. After finished each round, r will be incremented by one (line 20). Upon reaching the $\lceil (t+1)/2 \rceil$ th round, **Improved Algorithm 2** will exit the loop (line 21) and the host will make decision.

Corollary 2. The upper bound of the improved algorithm is $\lceil (t+1)/2 \rceil$.

Proof. By Theorem 1, if some hosts reaches the $\lceil (t+1)/2 \rceil$ th round, they must maintain the same value set. Letting these hosts make decision and terminate will not violate the properties of the uniform consensus. $\square$

Improved Algorithm 2 executed by a master agent

```

1. Initialization
2.  $V_i := \{v_i\}$ 
3.  $A_i := \emptyset$ 
4. for each  $CH_j$ , create_agent( $a_{ij}$ ), init( $a_{ij}$ ),  $A_i := A_i \cup \{a_{ij}\}$ 
5.  $r := 0$ ;
6. repeat
7.   for each  $a_{ij} \in A_i$ ,  $V_{ij} := V_i$ , signal( $a_{ij}$ )
8.   if itself is a Coordinating Host then
9.     wait_timeout( $T_s$ ) // all agents migrate to it can arrive
10.    signal all arrived agents
11.    wait_timeout( $T_s$ ) // its all agents can return
12.   else wait_timeout( $2T_s$ ) // its all agents can return
13.   end if
14.    $A_i := \{ \text{all agents that came back} \}$ 
15.   if  $A_i = \emptyset$  then  $f_i := \text{false}$ 
16.   else
17.      $V_i := \bigcap_{a_{ij} \in A_i} V_{ij}$ 
18.      $f_i := OR_{a_{ij} \in A_i} f_{ij}$ 
19.   end if
20.    $r := r + 1$ 
21. until  $f_i = \text{false}$  or  $r = \lceil (t+1)/2 \rceil$ 
22.  $A_i := \emptyset$  // delete all agents
23.  $d_i := \min(V_i)$ 

```

Recall that the notion “round” used in this paper is different from the existing early stopping consensus algorithms in synchronous system. The time cost of a round in our proposed algorithm is nearly the double of it those algorithms. In [11,8], the upper bound of an early stopping consensus algorithm is $\min(t+1, f+2)$, where t is the maximum number of failures can be tolerate and $f (f \leq t)$ is the number of failures actually occur. The upper bound of our improved algorithm is $\min(\lceil (t+1)/2 \rceil, \lceil (f+1)/2 \rceil + 1)$. For comparison, we convert it to the rounds of the existing early stopping consensus algorithms:

$$\lceil (t+1)/2 \rceil \times 2 = t+2 - t\%2 = t+1 + (1-t\%2), \text{ and}$$

$$(\lceil (f+1)/2 \rceil + 1) \times 2 = f+2 - f\%2 + 2 = f+2 + (2-f\%2).$$

We can see that for an upper bound defined on t , if t is odd, the upper bound is the same, but if it is even, the upper bound of our improved algorithm will be larger. For an upper bound defined on f , the upper bound of our improved algorithm will be larger. However, in our algorithm, a host can make decision and terminate immediately. It does not need to broadcast *decide* message but all existing early stopping consensus algorithms do this.

4.3 Worst Case Analysis

In the worst case, the improved algorithm will execute $\lceil (t+1)/2 \rceil$ rounds. This will happen in the case where there are total t crashes, and two host crashes occur in each foregoing round, one crash of a nonCH host in the first step and one of a CH in the second step. For example, h_k proposes v_k different from the value proposed by any other host. h_k dispatches agents to all CHs except CH_i – it crashed before it sends an agent to CH_i . Later, CH_i crashes after it lets all agents return home but before it sends out a_{li} . Consequently, at the end of round 1, all live hosts maintain the same value set not including v_k and *Altered_Flag* is true except h_l which contains v_k in its value set. The algorithm repeats the same execution pattern as in the first round until it reaches the $\lceil (t+1)/2 \rceil$ th round. In the $\lceil (t+1)/2 \rceil$ th round, at most one host can crash and, according to Lemma 3, all live hosts will maintain the same value set at the end of the round. Then all hosts can make decision and terminate.

According to Algorithm 2, a delay of $2T_s$ is the minimum required time for each round. Where T_s describes the maximum time a mobile agent requires for migrating from a host h_i to another host h_j . It consists of several time elements, as below.

$T_s = \max(\text{packing delay at } h_i + \text{migrate delay at link } + \text{unpacking delay at } h_j + \text{working delay at } h_j)$, where $1 \leq i, j \leq n$

So the total time cost in the worst case is:

$$T_{\text{worst}} = 2 \times \lceil (t+1)/2 \rceil T_s$$

To obtain the number of agent migrations, noting that the crash events will decrease the number of mobile agents after every round. In the worst case, we have the following scenario.

In the first round, there are $(t+1)n-1$ agents migrate to $t+1$ CHs, and one CH crashes and totally $(t+1)n-2$ agents returning but $2t+1$ agents can not find its home because the two crashed hosts in this round.

In the r th round, there will be $(t+1-r+1)(n-r+1)$ agents migrate to $t+1-r+1$ CHs and one CH crashes and totally $(t+1-r+1)(n-r+1)-1$ agents returning but $2(t-r+1)+1$ agents can not find its home because the two crashed hosts in this round.

Summing up these values, we have the number of migrations after round r :

$$D_{\text{worst}} = \sum_{i=1}^r (2(t+1-i+1)(n-i+1)-1) - 2$$

$$= \sum_{i=1}^r (2(t+2-i)(n+1-i)-1) - 2$$

4.4 Best Case Analysis

The best case for the algorithm occurs when t crashes happen in different t CHs immediately after the algorithm starts. This causes all mobile agents going to

those *CHs* being lost. Agents returning to those partitions also get lost. And every agent proposes the same value. As the result, only one round is needed and the number of migrations for the round is

$$D_{best} = n(t+1) + (n-t) = nt + 2n - t$$

Because the algorithm executes only one round, the maximum time delay is $2T_s$, $T_{best} = 2T_s$

5. Simulation and Evaluation

To demonstrate the effectiveness of the proposed mobile agent enabled algorithm, we have performed a simulation on the IBM Aglet mobile agent platform [17]. The Aglet is a Java-based mobile agent framework. The IBM Aglet platform also provides an aglet viewer called *Tahiti* and aglet servers, which are powerful machines that can host large number of aglets. It is easily to modeling objects in the algorithm, such as hosts, value sets, mobile agents, and underlying network system, etc.

In the simulation, we consider two types of costs: time and number of dispatched mobile agents. In the previous section, we have discussed the two types of costs for the best and the worst cases. The experiments aim to find out the costs in average cases. We used an even distribution for crashed of the hosts. In other words, the actual numbers of hosts that fail in a series of executions of the algorithm are evenly distributed in the closed interval $[0, t]$, where t is the maximum number of hosts that can fail.

Mobile agents are simulated as Java threads. The main thread of the simulation application is in responsible of injecting crashes. After initialization, it sleeps for a while, and randomly chooses a host object to make it “down” by calling its `crash()` method.

The following properties may affect the performance of the algorithm: (1) the *actual* number of hosts that crash, (2) the frequency of crash events. We analyzed two performance measures, namely, execution time and total number of agent migrations.

5.1 Number Of Migrations to Actual Crashes

The simulation environment consists of 20 hosts and up to 19 of them can crash. The number of actual crashes is in the range of $[0, 19]$, i.e., $t = 19$. The simulation is executed for 5000 times. This is enough to generate multiple instances for each possible number of actual crashes. Figure 2 shows the results we obtained from the executions.

In Figure 2, we plotted the graph of Number Of Migrations at Y-axis against Actual Crashes at X-axis.

We found that the more hosts crashed in the network, the less migrations would occur. This is reasonable because the crashes can destroy mobile agents. This leads to the decrease of total migrations.

Figure 2 – Number of Migrations against Actual Crashes

5.2 Execution Time to Actual Crashes

Figure 3 shows the average execution time as the function of the number of crashes.

Figure 3 – Execution Time (in milliseconds) vs. Actual Crashes

From the figure, we found that the execution time is not affected very much by the actual number of crashes – most time, the algorithm just needs two rounds for all hosts to make decision. This somewhat is different from our expectation. After analysis, we found the reason for this - during a round, the effect on the algorithm’s execution time by a single crash or multiple crashes within a partition is the same, and the probability of the occurrence of the worst case is too small.

Throughout the experiments, we understand how the performance is affected by the properties of the network. These properties are determined by the factors such as the number of actual failed host, and the frequency of the crashes.

6. Conclusions and Future Work

In this paper we have presented a mobile agent enabled consensus algorithm for synchronous distributed systems. The algorithm is designed within the MADSG framework where cooperating mobile agents are used as an aid to structuring distributed server groups. The correctness of the proposed algorithm is proved and the performance is evaluated by analysis and simulation.

We would like to emphasize that the algorithm proposed in this paper represent only a specific example of the MADSG algorithm family. Various extensions and improvements to the algorithm are possible and remain to be explored. Our future work includes improving the algorithm for synchronous distributed systems and design mobile agent enabled consensus algorithms for asynchronous distributed systems.

Acknowledgement

This work is partially supported by the University Grant Council of Hong Kong under the CERG Grant B-Q518 (PolyU 5076/01E) and the Hong Kong Polytechnic University, under HK PolyU ICRG grant A-P202.

References

- [1] M. K. Aguilera, W. Chen, and S. Toueg, "Failure Detection and Consensus in the Crash-Recovery Model", Technical Paper, 1999.
- [2] J. Cao, G. H. Chan, W. Jia, and T. Dillon, "Checkpointing and Rollback of Wide-area Distributed Applications Using Mobile Agents", Proc. IEEE 2001 International Parallel and Distributed Processing Symposium (IPDPS2001), San Francisco, USA, Apr. 2001.
- [3] J. Cao, T. S. Chan and J. Wu, "Achieving Replication Consistency using Cooperating Mobile Agents", Proc. IPPC 2001 Workshop on Wireless Networks and Mobile Computing (IEEE Computer Society Press), Valencia, Spain. Sep. 2001.
- [4] J. Cao, X. Wang and S. K. Das, "A Framework of Using Cooperating Mobile Agents to Achieve Load Sharing in Distributed Web Server Groups", to appear in 5th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP2002), Beijing, China, Oct., 2002.
- [5] J. Cao, X. Wang and J. Wu, "A Mobile Agent Enabled Fully Distributed Mutual Exclusion Algorithm", to appear in *Proc. IEEE 6th International Conference on Mobile Agents*, Barcelona, Spain. Oct 2002.
- [6] T. Chandra and S. Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems", J. ACM, vol. 43, no. 2, Mar. 1996, pp. 225-267.
- [7] B. Charron-Bost, R. Guerraoui, and A. Schiper, "Synchronous system and perfect failure detector: solvability and efficiency issues", Proc. IEEE Int. Conf. on Dependable Systems and Networks (DSN), New York, USA, June 2000, pp. 523-532.
- [8] B. Charron-Bost and A. Schiper, "Uniform consensus harder than consensus", Technical Report DSC/2000/028, École Polytechnique Fédérale de Lausanne, Switzerland, May 2000.
- [9] B. Chor, M. Merritt, and D. B. Shmoys, "Simple constant-time consensus protocols in realistic failure models", J. ACM, vol. 36, no. 3, Jul. 1989, 591-614.
- [10] F. Cristian, and C. Fetzer, "The Timed Asynchronous Distributed System Model", IEEE Transactions on Parallel and Distributed Systems, vol. 10, no. 6, Jun. 1999, 642-657.
- [11] D. Dolev, R. Reischuk, and R. Strong, "Early Stopping in Byzantine Agreement", J. ACM, vol. 37, no. 4, Apr. 1990, 720-741.
- [12] C. Dwork, and N. Lynch, "Consensus in the Presence of Partial Synchrony", J. ACM, vol. 35, no. 2, Apr. 1988, 288-323.
- [13] M.J. Fischer, N. Lynch, and M.S. Paterson, "Impossibility of Distributed Consensus with One Faulty Process", J. ACM, vol. 32, no. 2, Apr. 1985, 374-382.
- [14] R. S. Gray, G. Cybenko, D. Kotz, and D. Rus. "Mobile agents: Motivations and State of the Art", In Jeffrey Bradshaw, editor, Handbook of Agent Technology, AAAI/MIT Press, 2001.
- [15] R. Guerraoui, M. Hurfin, A. Mostefaoui, R. Oliveira, M. Raynal, and A. Schiper, "Consensus in Asynchronous Distributed Systems: A Concise Guided Tour", In S. Shrivastava S. Krakowiak, editor, Advances in Distributed Systems, number LNCS 1752, Springer, 2000, 33-47.
- [16] V. Hadzilacos and S. Toueg, "Reliable Broadcast and Related Problems", Distributed Systems, S. Mullender ed., 1993, 97-145.
- [17] D.B. Lange and M. Oshima, "Programming And Deploying Java Mobile Agents With Aglets", Addison Wesley, 1998.
- [18] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem", ACM Transactions on Programming Languages and Systems, Jul. 1982, 382-401.
- [19] N. Lynch, "Distributed Algorithms", Morgan Kaufmann, 1996.
- [20] D. B. Lange and M. Oshima, "Seven Good Reasons for Mobile Agents", *Communication of the ACM*, Vol. 42, No. 3, March 1999. pp. 88-89.
- [21] N. Minar, K.H. Kramer and P. maes, "Cooperative Mobile Agents for Dynamic network Routing", in *Software Agents for Future Communication Systems*, Springer-Verlag, 1999.
- [22] M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults", J. ACM, vol. 27, no. 2, Apr. 1980, 228-234.