CS 3551 DISTRIBUTED COMPUTING

UNIT IV CONSENSUS AND RECOVERY **10**

Consensus and Agreement Algorithms: Problem Definition – Overview of Results – Agreement in a Failure-Free System(Synchronous and Asynchronous) – Agreement in Synchronous Systems with Failures; Checkpointing and Rollback Recovery: Introduction – Background and Definitions – Issues in Failure Recovery – Checkpoint-based Recovery – Coordinated Checkpointing Algorithm -- Algorithm for Asynchronous Checkpointing and Recovery

Consensus (General Agreement)

IV → dest

DB Commit

P_1 P_2 P_3 P_4

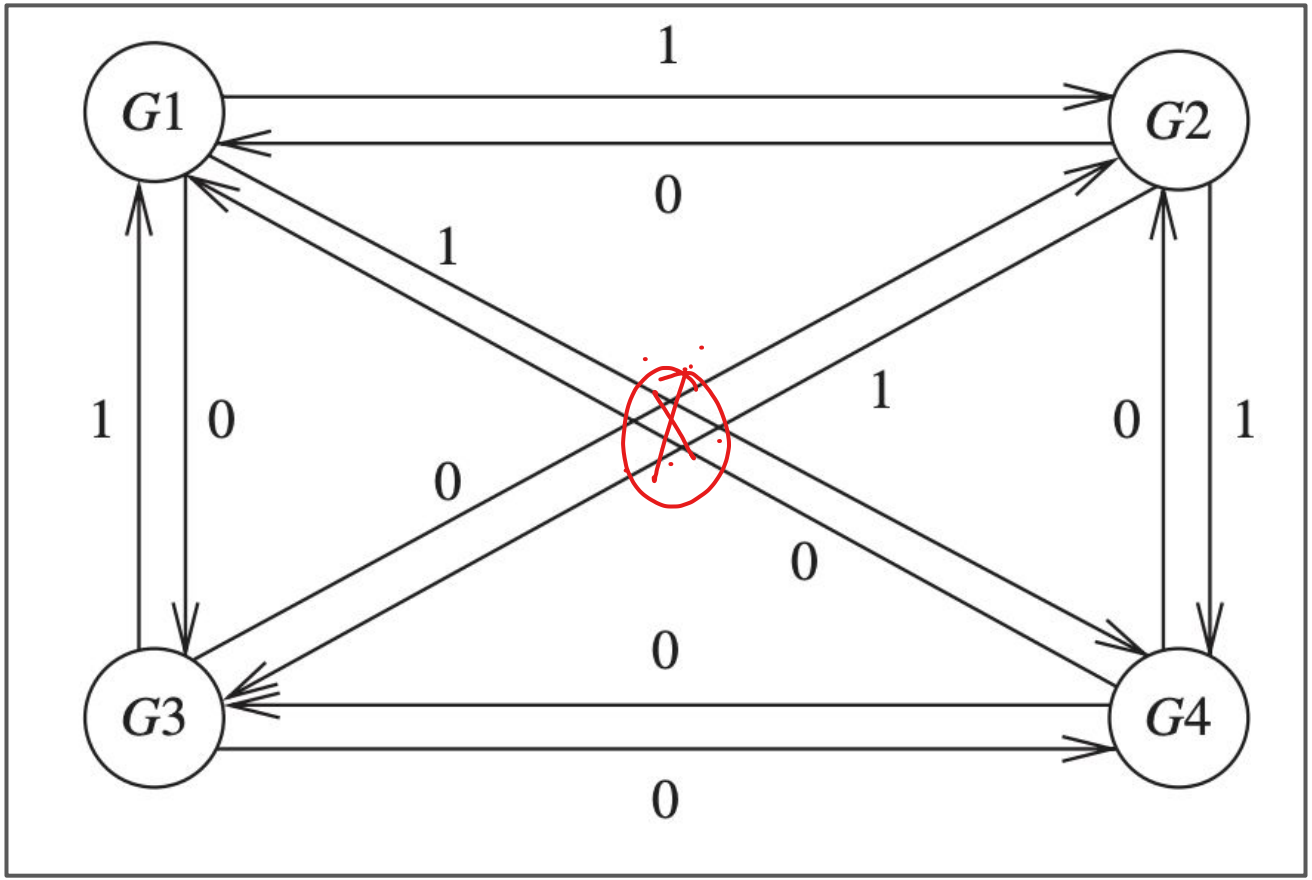
Assumptions

1. **Failure models**
   a. Among the n processes in the system, at most f processes can be faulty.
   b. A faulty process can behave in any manner allowed by the failure model assumed.
   c. In fail-stop model, a process may crash in the middle of a step, which could be the execution of a local operation or processing of a message for a send or receive event.
   d. In the Byzantine failure model, a process may behave arbitrarily
2. **Synchronous/asynchronous communication**
   a. If a failure-prone process chooses to send a message to process Pi but fails, then Pi cannot detect the non-arrival of the message in an asynchronous system
   b. In synch systems, message which has not been sent can be recognized by the intended recipient, at the end of the round.
3. **Network connectivity**
   a. The system has full logical connectivity, i.e., each process can communicate with any other by direct message passing.
4. **Channel reliability**
   a. The channels are reliable, and only the processes may fail.

Assumptions continued…

5. **Sender identification**
   a. A process that receives a message always knows the identity of the sender process.
   b. When multiple messages are expected from the same sender in a single round, we implicitly assume a scheduling algorithm that sends these messages in sub-rounds, so that each message sent within the round can be uniquely identified.
6. **Authenticated vs. non-authenticated messages**
   a. With unauthenticated messages, when a faulty process relays a message to other processes, (i) it can forge the message and claim that it was received from another process, and (ii) it can also tamper with the contents of a received message before relaying it.
   b. When a process receives a message, it has no way to verify its authenticity. An unauthenticated message is also called an **oral message** or an unsigned message.
   c. Using authentication via techniques such as digital signatures, it is easier to solve the agreement problem.
7. **Agreement variable**
   a. The agreement variable may be boolean or multivalued, and need not be an integer

Byzantine Empire



Case Study

- Four camps of the attacking army, each commanded by a general, are camped around the fort of Byzantium.
- They can succeed in attacking only if they attack simultaneously. Hence, they need to reach agreement on the time of attack.
- The only way they can communicate is to send messengers among themselves.
- The messengers model the messages. An asynchronous system is modeled by messengers taking an unbounded time to travel between two camps.
- A lost message is modeled by a messenger being captured by the enemy. A Byzantine process is modeled by a general being a traitor.
- The traitor will attempt to subvert the agreement-reaching mechanism, by giving misleading information to the other generals. For example, a traitor may inform one general to attack at 10 a.m., and inform the other generals to attack at noon. Or he may not send a message at all to some general.
- Likewise, he may tamper with the messages he gets from other generals, before relaying those messages.
- The various generals are conveying potentially misleading values of the decision variable to the other generals, which results in confusion. In the face of such Byzantine behavior, the challenge is to determine whether it is possible to reach agreement, and if so under what conditions.
- If agreement is reachable, then protocols to reach it need to be devised.

LIKE

COMMENT

SHARE

SUBSCRIBE

① The Byzantine agreement problem

P_1
$x = 10$

P_2
P_3
P_4

P_5 – faulty

The Byzantine agreement problem requires a designated process, called the source process, with an initial value, to reach agreement with the other processes about its initial value,

- **Agreement** All non-faulty processes must agree on the same value.
- **Validity** If the source process is non-faulty, then the agreed upon value by all the non-faulty processes must be the same as the initial value of the source.
- **Termination** Each non-faulty process must eventually decide on a value.

② The consensus problem

The consensus problem differs from the Byzantine agreement problem in that each process has an initial value and all the correct processes must agree on a single value [20, 25]. Formally:

- **Agreement** All non-faulty processes must agree on the same (single) value.
- **Validity** If all the non-faulty processes have the same initial value, then the agreed upon value by all the non-faulty processes must be that same value.
- **Termination** Each non-faulty process must eventually decide on a value.

P1 1 [1 4 8] P2 4 [1 4 8] P3 8 [1 4 8]

3 The interactive consistency problem

The interactive consistency problem differs from the Byzantine agreement problem in that each process has an initial value, and all the correct processes must agree upon a set of values, with one value for each process [20, 25]. The formal specification is as follows:

- **Agreement** All non-faulty processes must agree on the same array of values $A[v_1 \ldots v_n]$.
- **Validity** If process i is non-faulty and its initial value is v_i, then all non-faulty processes agree on v_i as the ith element of the array A. If process j is faulty, then the non-faulty processes can agree on any value for $A[j]$.
- **Termination** Each non-faulty process must eventually decide on the array A.

Agreement in a failure-free system (synchronous or asynchronous)

- In a failure-free system, consensus can be reached by
  - collecting information from the different processes
  - arriving at a “decision,”
  - distributing this decision in the system.
- A distributed mechanism would have each process broadcast its values to others, and each process computes the same function on the values received.
- The decision can be reached by using an application specific function – some simple examples being the majority, max, and min functions.
- Algorithms to collect the initial values and then distribute the decision may be based on the token circulation on a logical ring, or the three-phase tree-based broadcast–convergecast–broadcast, or direct communication with all nodes.
- In a synchronous system, this can be done simply in a constant number of rounds.
- In an asynchronous system, consensus can similarly be reached in a constant number of message hops.

LIKE

COMMENT

SHARE

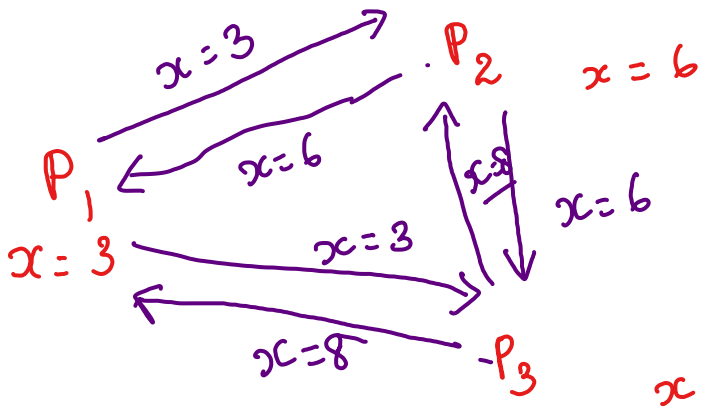
SUBSCRIBE

Agreement in (message-passing) synchronous systems with failures

1. Consensus algorithm for crash failures (synchronous system)
2. Upper bound on Byzantine processes
3. Byzantine agreement tree algorithm: exponential (synchronous system)
   a. Recursive
   b. Iterative
4. Phase-king algorithm for consensus:

1. Consensus Algorithm for Crash Failures

1-round



@P1

min(3, 6, 8) => 3

x=3

@P2

min(6, 3, 8) = 3

x=3

@P3

min(8, 3, 6) = 3

x=3

(global constants)
integer: f; // maximum number of crash failures tolerated
(local variables)
integer: $x \longleftarrow$ local value;

(1) Process P_i $(1 \leq i \leq n)$ executes the consensus algorithm for up to f crash failures:
(1a) **for** *round* **from** 1 **to** $f+1$ **do**
(1b) **if** the current value of x has not been broadcast **then**
(1c) **broadcast**(x);
(1d) $y_j \longleftarrow$ value (if any) received from process j in this round;
(1e) $x \longleftarrow min_{\forall j}(x, y_j)$;
(1f) **output** x as the consensus value.

Algorithm 14.1 Consensus with up to f fail-stop processes in a system of n processes, $n > f$ [8]. Code shown is for process P_i, $1 \leq i \leq n$.

Validity of Conditions

- The *agreement condition* is satisfied because in the $f+1$ rounds, there must be at least one round in which no process failed. In this round, say round r, all the processes that have not failed so far succeed in broadcasting their values, and all these processes take the minimum of the values broadcast and received in that round. Thus, the local values at the end of the round are the same, say x_i^r for all non-failed processes. In further rounds, only this value may be sent by each process at most once, and no process i will update its value x_i^r.
- The *validity condition* is satisfied because processes do not send fictitious values in this failure model. (Thus, a process that crashes has sent only correct values until the crash.) For all i, if the initial value is identical, then the only value sent by any process is the value that has been agreed upon as per the *agreement condition*.
- The *termination condition* is seen to be satisfied.

Time Complexity

There are $f+1$ rounds, where $f < n$. The number of messages is at most $O(n^2)$ in each round, and each message has one integer. Hence the total number of messages is $O((f+1) \cdot n^2)$.

Key Points

Algorithm 14.1 gives a consensus algorithm for n processes, where up to f processes, where $f < n$, may fail in the fail-stop model [8]. Here, the consensus variable x is integer-valued. Each process has an initial value x_i. If up to f failures are to be tolerated, then the algorithm has $f+1$ rounds. In each round, a process i sends the value of its variable x_i to all other processes if that value has not been sent before. Of all the values received within the round and its own value x_i at the start of the round, the process takes the minimum, and updates x_i. After $f+1$ rounds, the local value x_i is guaranteed to be the consensus value.

LIKE

COMMENT

SHARE

SUBSCRIBE