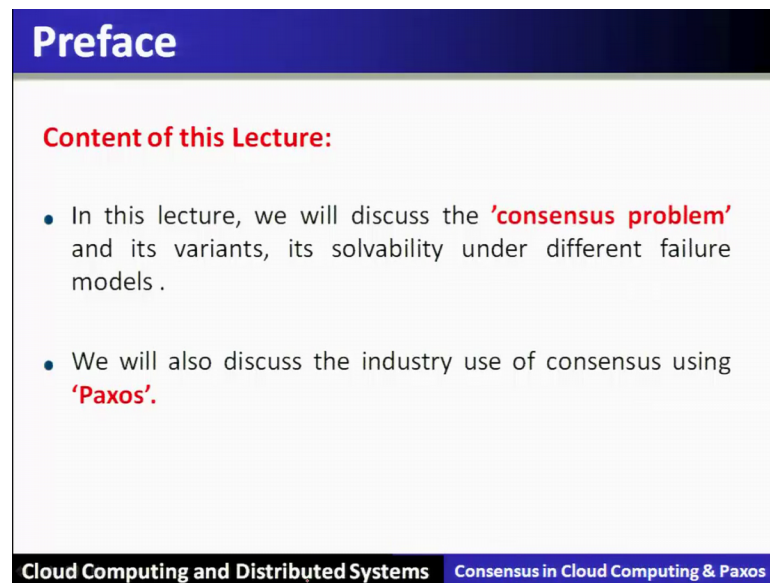


Cloud Computing and Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture - 13
Consensus in Cloud Computing and Paxos

Consensus in Cloud Computing and Paxos protocol.

(Refer Slide Time: 00:17)



Preface

Content of this Lecture:

- In this lecture, we will discuss the '**consensus problem**' and its variants, its solvability under different failure models .
- We will also discuss the industry use of consensus using '**Paxos**'.

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

Preface content of this lecture. We will cover consensus problem and its variants. It is solvability under different failure models. We will discuss in this part of a lecture. We will also discuss the industry variant of consensus using a well-known protocol which is called Paxos.

(Refer Slide Time: 00:41)

Common issues in consensus

Consensus problems in DS & name Reliability - 5 factors

Consider a group of servers attempting together:

- ✓ Make sure that all of them receive the same updates in the same order as each other **[Reliable Multicast]**
- ✓ To keep their own local lists where they know about each other, and when anyone leaves or fails, everyone is updated simultaneously **[Membership/Failure Detection]**
- ✓ Elect a leader among them, and let everyone in the group know about it **[Leader Election]**
- ✓ To ensure mutually exclusive (one process at a time only) access to a critical resource like a file **[Mutual Exclusion]**

99.99999%
100% reliability
System in face of failure
Consensus

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

Let us understand what a consensus is. So, before going ahead let us see the importance of this problem in cloud and distributed computing.

So, in a cloud computing different vendors they provide the reliability support for the services. The reliability services ensures that it is, let us say 5 number of factors such as 99.99999 percentage of reliability; that means, it is not possible to provide 100 percent reliable system in the face of failures. So, that means, if the underlying systems are the servers are failing how about the services reliability. So, none of the vendors or none of them can provide 100 percent reliability in their services; however, they support the reliabilities, and it is mentioned that 99 point how many number of 9's; that means, how reliable the system is, but it is not 100 percent reliable.

Now, the this particular reliability how they are ensuring that is achieved through the consensus protocols. So, that means, when a group of servers are attempting together to solve a particular problem in spite of failure they have to reach to a consensus. And therefore, they achieve this kind of solutions. But this is not so trivial it is very, very complex in different models that we will cover in this part of discussion.

So, let us understand where this consensus problems lies and what are their names in the distributed systems. So, let us consider a group of servers who are attempting together to make sure that all of them receives the same updates in the same order as they send. So,

this kind of problem is known as reliable multicast problem. And this requires the solution of consensus to be involved.

The second kind of problem is that to keep their own local list; where they know about each other and when they and when anyone leaves or fails. So, everyone updates simultaneously. So, this kind of problem is called membership problem or a failure detection problem. This also requires the consensus or this particular problem is also equivalent to the consensus, that we will see the later part.

Similarly, a group of servers who are attempting together to elect a leader among them, and this particular decision of electing the leader is known to everyone in that group is called a leader election problem. So, leader election under this particular failure requires the solution of consensus. So, if you can solve the consensus problem then you can solve the leader election, and that is why if they are all equivalent problems.

Similarly there is another problem which is considering that a group of servers attempting together to ensure the mutually exclusive access to the critical resources, such as files writing on a file and so on or an update on the replicated database. Which requires access to the competing access to one process among the competing processes at a particular instant of time. So, that is this problem is called as a mutual exclusion problem. In the failure this problem is also requires the consensus protocols for reliability aspect.

(Refer Slide Time: 06:21)

So what is common?

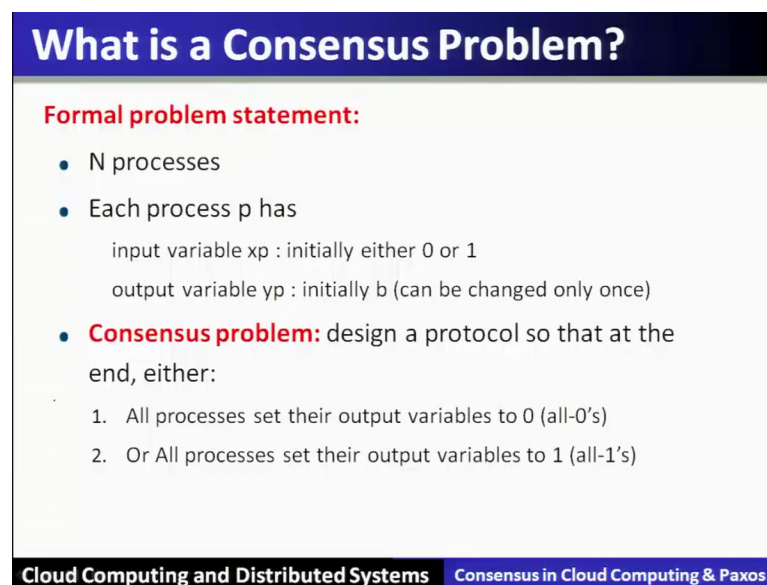
- All of these were groups of processes attempting to **coordinate** with each other and reach **agreement** on the value of something
 - The ordering of messages
 - The up/down status of a suspected failed process
 - Who the leader is
 - Who has access to the critical resource
- All of these are related to the **Consensus** problem

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

So, let us see that what is common in all these problems. So, here we have seen that all of these group of processes, they are attempting to coordinate with each other and reach an agreement on something in different problems that we have seen leader election mutual exclusion and so on. So, the agreement on that values depends upon different problems, let us understand what kind of agreement is required to solve this consensus problem.

So, it depends upon whether it is an ordering of messages at all the processes in the groups the up or down status of the suspected field process and who is a leader and who has the access to the critical resource. So, that depends upon different values to be agreed upon to solve this particular these kind of problems. All of these are related to the consensus problem.

(Refer Slide Time: 07:23)



What is a Consensus Problem?

Formal problem statement:

- N processes
- Each process p has
 - input variable x_p : initially either 0 or 1
 - output variable y_p : initially b (can be changed only once)
- **Consensus problem:** design a protocol so that at the end, either:
 1. All processes set their output variables to 0 (all-0's)
 2. Or All processes set their output variables to 1 (all-1's)

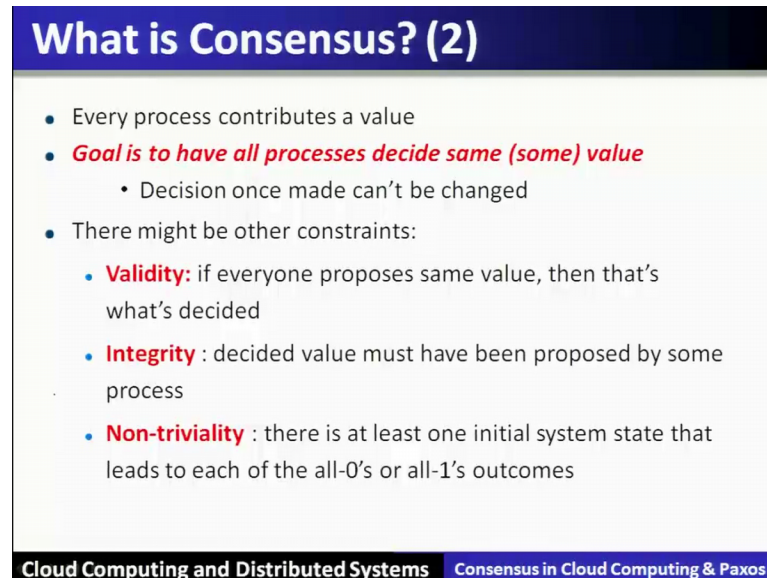
Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

So, having known what a consensus is. So, let us formally define what a consensus problem is for our discussion in this lecture.

So, the formally statement says that, let us have a system of N processes where each process has the input variable x_p . Initially it can have the values either 0 or 1 in that variable x_p . Similarly, the output variable y_p initially which is assigned to b which can be changed only once and once output means, whatever value is assigned it will not be changed then after the end of consensus algorithm. So, consensus problem says to design a protocol. So, the at the end either all the processes in that particular group. They set

their output variables to 0s; that means, all 0's or the processes sets their output variables to 1, that is all 1's. Then only this particular statement is called the consensus problem.

(Refer Slide Time: 08:40)



What is Consensus? (2)

- Every process contributes a value
- **Goal is to have all processes decide same (some) value**
 - Decision once made can't be changed
- There might be other constraints:
 - **Validity**: if everyone proposes same value, then that's what's decided
 - **Integrity** : decided value must have been proposed by some process
 - **Non-triviality** : there is at least one initial system state that leads to each of the all-0's or all-1's outcomes

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

Now, here in this particular consensus problem, every process will contribute toward value and the goal is to have all the processes decide on some value. So, once the decision is finalized it cannot be changed. There are some other constraints let us see, which will be required to be enforced to solve this particular consensus problem. The first constraint is called validity that everyone proposes same value, then that is what is being decided called validity.

Integrity means that decided value must have been proposed by some process. Non-triviality says that there is at least one initial state that leads to each of the all 0's or all ones outcomes. So, 3 different properties or constraints they have to be satisfied. Then only we can say that the consensus problem is solved validity integrity non-triviality.

(Refer Slide Time: 09:57)

Why is it Important?

- Many problems in distributed systems are **equivalent to (or harder than)** consensus!
 - Perfect Failure Detection
 - Leader election (select exactly one leader, and every alive process knows about it)
 - Agreement (harder than consensus)
- So consensus is a very important problem, and solving it would be really useful!
- **So, is there a solution to Consensus?**

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

Why this consensus problem is important? So, here we can see in many problems in the distributed systems they are equivalent to the consensus problem or even harder than that; that means, if you can solve consensus problem those problems in the distributed systems are also been solved.

For example, failure detection. So, if you can detect a failure; that means, you can solve the consensus problem. Or consensus problem if it is, solvable then you can also detect the failures similarly leader election exactly one leader, and every alive process knows about it called leader election problem, if leader election problem is equivalent to the consensus problem. Similarly, the agreement problem; that means, they can reach all of them reach to an agreement to a common value is harder than consensus problem.

So, the consensus is very important problem and solving it would be really useful. So, is there any solution to a consensus problem we will see that it is very, very complex problem. It is not trivial to solve this particular problem in some of the models.

(Refer Slide Time: 11:08)

Two Different Models of Distributed Systems

- **Different Models of Distributed Systems:**
 - (i) **Synchronous System Model and**
 - (ii) **Asynchronous System Model**

(i) Synchronous Distributed System

- Each message is received within bounded time
- Drift of each process' local clock has a known bound
- Each step in a process takes $lb < \text{time} < ub$

E.g., A collection of processors connected by a communication bus, e.g., a Cray supercomputer or a multicore machine

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

So, there are 2 different models of distributed system which also is applied to the cloud computing systems. So, we will see under these models how the consensus problem can be solved.

So, the first of these 2 models is called synchronous distributed system. So, synchronous model says that each message is received within a bounded time. And the drift of each processes local clock is also bounded. And each step in a process is bounded in a time between the lower and upper bounds. So, meaning in the sense that in the synchronous distributed system, the assumptions are being made that each process is delivered within a bounded period of time and the processes local clocks also when they drift that also is bounded and each step in a process is also bounded then only it is called synchronous distributed systems.

(Refer Slide Time: 12:25)

Asynchronous System Model

(ii) Asynchronous Distributed System

- No bounds on process execution
- The drift rate of a clock is arbitrary
- No bounds on message transmission delays

E.g., The Internet is an asynchronous distributed system, so are ad-hoc and sensor networks

- This is a more **general (and thus challenging)** model than the synchronous system model.
- A protocol for an asynchronous system will also work for a synchronous system (but not vice-versa)

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

Asynchronous on the other hand, does not have any bounds specified on the process execution. Similarly, the clocks when they drift that particular bound is also arbitrary and the message transmission delays; that means, when the message is received, that particular delay also is not bounded then that model is called asynchronous distributed or asynchronous model. Internet is a perfect example of asynchronous distributed system model and there are many other networks which is an example of a asynchronous model.

So, there is a more general model than the synchronous system model; that means, if you can solve the problem under this model asynchronous system model. That means, that problem also can be solved in a synchronous model, but vice versa is not valid.

(Refer Slide Time: 13:35)

Possible or Not

- In the synchronous system model:
 - Consensus is solvable
- In the asynchronous system model:
 - Consensus is impossible to solve
 - Whatever protocol/algorithm you suggest, there is always a worst-case possible execution (with failures and message delays) that prevents the system from reaching consensus
 - Powerful result (*FLP proof*)
 - Subsequently, **safe or probabilistic solutions** have become quite popular to consensus or related problems.

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

So, let us see the consensus problem in this the 2 models scenarios. So, if the model is synchronous distributed system, then consensus is solvable that is being proved and we will see in this part of the lecture.

However, in asynchronous system model the consensus is impossible to solve, that is also being proved. So, that means, that whatever protocol algorithms, you have there is always a worst possible execution scenario with the failures and the message delays, that will prevent the system from reaching the consensus. That is why the consensus is impossible to solve. And this was proved in FLP a famous proof of this FLP we will see has proved that consensus in a asynchronous system is impossible.

So, using this particular results safe or probabilistic solutions to this particular problem have become quite popular to the consensus problem, because it has a lot of use.

(Refer Slide Time: 14:53)

FLP

- One of the most important results in distributed systems theory was published in April 1985 by Fischer, Lynch and Patterson (FLP)
- Their short paper 'Impossibility of Distributed Consensus with One Faulty Process', which won the Dijkstra award, placed an upper bound on what it is possible to achieve with distributed processes in an asynchronous environment.

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

We will see all these. Let us see what FLP is, it is one of the most important results in a distributed system theory, published in April 1985 by Fisher Lynch and Patterson. That is called FLP problem. So, this particular problem, you can refer to that paper which is called as impossibility of distributed, consensus with one faulty process; which won the Dijkstra award, and has shown an upper bound on what it is possible to achieve with the distributed processes in asynchronous environment that we will discuss.

(Refer Slide Time: 15:40)

Let's Try to Solve Consensus!

System model (Assumptions):

- **Synchronous system:** bounds on
 - Message delays
 - Upper bound on clock drift rates
 - Max time for each process stepe.g., multiprocessor (common clock across processors)
- Processes can fail by stopping (crash-stop or crash failures)

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

So, with this particular background that, the system model plays a important role whether you can solve the consensus or not. So, we have to study the algorithms accordingly. So, we will first assume the system model to be the synchronous system model, and also we will assume that there exist a bound on the message delays. And they are exist an upper bound on clock drift rates. And there also exist the bound on each processes steps to complete in the model. And that is called synchronous system model that we will assume. We will also assume a failure model which is called a crash fault or a crash failure. This model assumes that the process when fails they will stop that is called crash fault model of failure.

So, with these 2 assumptions that we will assume a synchronous system model and a crash fault system model let us see how we are going to solve the consensus in this model.

(Refer Slide Time: 17:02)

Consensus in Synchronous Systems

- **For a system with at most f processes crashing**
 - All processes are synchronized and operate in “**rounds**” of time.
 - The algorithm proceeds in $f+1$ rounds (with timeout), using reliable communication to all members
 - $Values_i^r$: the set of proposed values known to p_i at the beginning of round r .

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

So, the consensus in this particular model, we are assuming that this system can have at most f processes crashing. So, there is a limit that is called f processes which can crash. All the processes are synchronized operates in the rounds of time. By round we mean that if the messages send it will be delivered when before the next round starts.

So, the algorithm proceeds in f plus 1 rounds using a reliable communication to all the members. Now here in this round every round we will, every process will exchange the values with other processes which is denoted as values of process i at the round r . So, the

set of proposed values known to a process i at the beginning of round r is denoted by this particular variable and this variable will be communicated through the multicast among the group who are participating in this consensus problem.

(Refer Slide Time: 18:26)

Consensus in Synchronous System

Possible to achieve!

- For a system with at most f processes crashing
 - All processes are synchronized and operate in "rounds" of time
 - the algorithm proceeds in $f+1$ rounds (with timeout), using reliable communication to all members.
 - $Values_i^r$: the set of proposed values known to p_i at the beginning of round r .

- Initially $Values_i^0 = \{\}$; $Values_i^1 = \{v_i\}$

```

for round = 1 to f+1 do
  multicast ( $Values_i^r$  -  $Values_i^{r-1}$ ) // iterate through processes, send each a message
   $Values_i^{r+1} \leftarrow Values_i^r$ 
  for each  $V_j$  received
     $Values_i^{r+1} = Values_i^{r+1} \cup V_j$ 
  end
end
 $d_i = \text{minimum}(Values_i^{f+1})$  // consistent minimum based on say, id (not minimum value)
  
```

Cloud Computing and Distributed Systems
Consensus in Cloud Computing & Paxos

So, let us see the algorithm for this. So, again these particular values which are known to the process i at the beginning of the round r is to be communicate (Refer Time: 18:43). So, let us see that initialize this particular value at the 0th round at the beginning of the 0th round as empty. And for the round number one the process i will have the value V_i . So, let us start for round 1 to round f plus 1. Because this particular since we have assumes that there are at most f different processes which can crash. So, algorithm has to take f plus 1 rounds to converge into the consensus problem.

So, let us understand the round number 1. So, round number 1 the process i will have it is value V_i and it will be multi casted. So, this particular values will be received and will be sent to all the processes and it will also receive the same values in the in that particular round 1 from the other processes let us say V_j . So, the values which is received out of that V_j will also be taken into that particular set of values, which will be collected at the end of round r that is here in this case.

So, when f plus 1 rounds finishes, then these set of values which is received by a process i it will take the minimum of all the values. So, every process like V_i , they will take the

minimum, and this particular protocol guarantees that all the processes will have the same value that is d at the end of $f+1$ rounds.

(Refer Slide Time: 20:54)

Why does the Algorithm work?

- After $f+1$ rounds, all non-faulty processes would have received the same set of Values. Proof by contradiction.
- Assume that two non-faulty processes, say p_i and p_j , differ in their final set of values (i.e., after $f+1$ rounds)
- Assume that p_i possesses a value v that p_j does not possess. (w.l.g.)
 - p_i must have received v in the very last round ($f+1$ round)
 - Else, p_i would have sent v to p_j in that last round
 - So, in the last round: a third process, p_k , must have sent v to p_i , but then crashed before sending v to p_j .
 - Similarly, a fourth process sending v in the last-but-one round must have crashed; otherwise, both p_k and p_j should have received v .
 - Proceeding in this way, we infer at least one (unique) crash in each of the preceding rounds.
 - This means a total of $f+1$ crashes, while we have assumed at most f crashes can occur => contradiction.

p_i & p_j - Same value & reach
 first

Cloud Computing and Distributed Systems
Consensus in Cloud Computing & Paxos

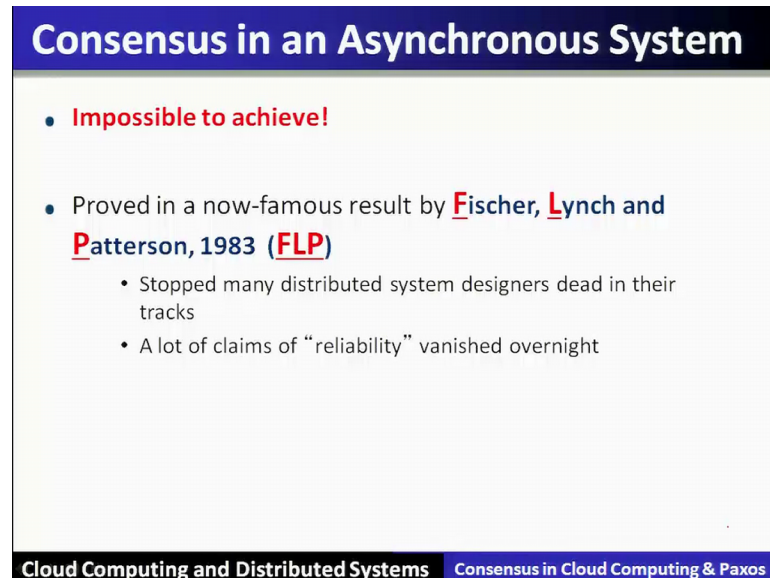
Now, let us see why this algorithm will work to solve the consensus problem. So, after $f+1$ rounds all the non-faulty processor or a processes would have received the same set of values. And we will see that particular proof by the contradiction.

So, let us assume that they are exists 2 non faulty processes. Let us say p_i and p_j and they differ in their final set of values. Let us say that after $f+1$ rounds that they have different values. Now assume that process p_i has the value v without loss of generality. And that p_j does not possess this value. It may have a different value, we can change it that p_j will have this vale, but p_i does not have that we have already assumed that without loss of generality.

Now, if p_i must have received the v in the very last round, that is $f+1$ th round, else p_i would have send value v to p_j in that last round. So, in this last round a third process p_k must have send v to p_i , but then crashed before sending v to p_j . Similarly, a 4th process sending v in the last, but oneth round must have crashed otherwise both p_k and p_j should have received v . Proceeding in this way we infer that at least one unique crash in each of the preceding round. This means that a total $f+1$ crashes, which will prevent the value v to be reached at p_j . While we have already assumed that there can be at most f crashes.

Therefore, this is a contradiction and this contradiction will say that p_i and p_j should have received the same value and they will reach the consensus.

(Refer Slide Time: 23:33)



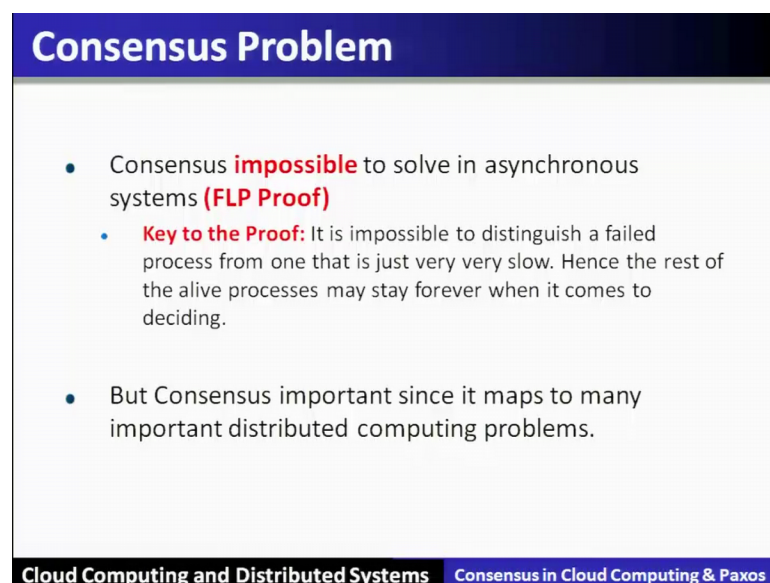
Consensus in an Asynchronous System

- **Impossible to achieve!**
- Proved in a now-famous result by **Fischer, Lynch and Patterson, 1983 (FLP)**
 - Stopped many distributed system designers dead in their tracks
 - A lot of claims of “reliability” vanished overnight

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

However, this consensus if you change the system model, asynchronous model it will become impossible to achieve. And we have already discussed that FLP that is Fischer Lynch and Patterson have already proved this particular theorem, that is called FLP theorem.

(Refer Slide Time: 23:58)



Consensus Problem

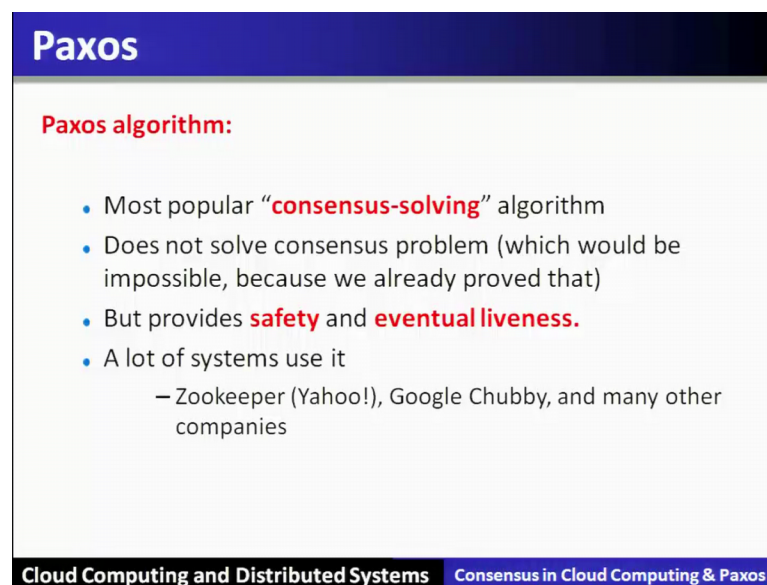
- Consensus **impossible** to solve in asynchronous systems (**FLP Proof**)
- **Key to the Proof:** It is impossible to distinguish a failed process from one that is just very very slow. Hence the rest of the alive processes may stay forever when it comes to deciding.
- But Consensus important since it maps to many important distributed computing problems.

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

So, consensus problem we have seen that this consensus impossible to be solved in asynchronous system model. And that is given by FLP proof. We will not going to the proof, but let us see what is the key issue which led to this particular proof. Now it is impossible to distinguish a failed process from that is just very, very slow. Hence, the rest of the alive process may stay forever when it comes to the deciding. So, that becomes a proof; that means, a failure or a system becoming very slow it is very difficult to detect it. Quickly it may take very long time to conclude it, and this will lead to that impossibility result that is called FLP.

Now, since consensus is are important problem. Since it maps to many important distributed computing problem. Therefore, we will see that in this asynchronous system model how we are going to solve the consensus.

(Refer Slide Time: 25:26)



Paxos

Paxos algorithm:

- Most popular “**consensus-solving**” algorithm
- Does not solve consensus problem (which would be impossible, because we already proved that)
- But provides **safety** and **eventual liveness**.
- A lot of systems use it
 - Zookeeper (Yahoo!), Google Chubby, and many other companies

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

So, once a solution is called Paxos algorithm. This Paxos algorithm is the most popular consensus solving algorithm. In the asynchronous system model; however, we have already seen that consensus in the asynchronous is unsolvable using FLP theorem, but as far as the Paxos is concerned, Paxos provides or solves to some extent the consensus problem in the asynchronous system model. But it provides the safety and eventual liveness, it is not the guaranteed liveness, but it is an eventual liveness.

So, the Paxos algorithm solves the consensus using these 2 constraints safety and eventual liveness. A lot of systems which are already deployed by the industry like

Yahoo and Google, they are using this algorithm Paxos algorithm in the form of Zookeeper. And also the Google stack that is called Chubby and many other companies also use the Paxos. So, that is why the Paxos is the most popular in solving the consensus.

(Refer Slide Time: 27:02)

Paxos

- Paxos is invented by **Leslie Lamport**
- Paxos provides **safety** and **eventual liveness**
 - **Safety:** Consensus is not violated – *Non-faulty processes agree not with same value. (everything good happens)* p_i..p_j.. → 0 (or 1)
 - **Eventual Liveness:** If things go well sometime in the future (messages, failures, etc.), there is a good chance consensus will be reached. But there is no guarantee.
- FLP result still applies: Paxos is not **guaranteed** to reach Consensus (ever, or within any bounded time)

Cloud Computing and Distributed Systems
Consensus in Cloud Computing & Paxos

Paxos is invented by the Leslie Lamport. He is a famous scientist working in Microsoft. And he has the winner of Turing award and several awards for his contribution in the distributed systems. So, he is basically the key to invent this Paxos system. So, Paxos will provide the safety and eventual liveness. So, safety says that consensus is not violated. Meaning to say that, if 2 processes which are non-faulty, they arrive at the decision with the same value.

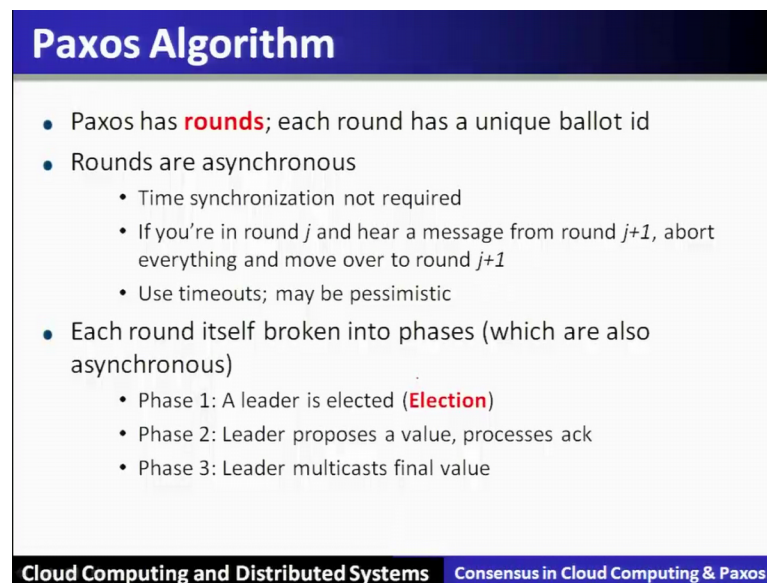
That is called safety in the consensus; that means, that non faulty processes reach agreement with the same value; that means, if p_i and p_j. So, at the end they may decide on the same value 0 or let us say 1, but not the scenario whether p_i will receive or p_i will decide 0 and p_j will receive 1. So, both of them either they decide on 0 or all of them decides on 1.

That means, the consensus is not violated and this ensures that Paxos supports or provides the safety or guarantees the safety properties the other property which is called eventual liveness. So, safety says that everything good happens. And eventual liveness says that if the things go well sometime in the future; that means, the messages failures

etcetera, then there is a good chance that consensus will be reached, but it is not guaranteed in this particular situation. That is sometimes a situation may arise that this liveness that eventually everything good happens is not guaranteed by the Paxos.

So, the FLP is still applied why because it is not the liveness, but it is an eventual liveness. So, Paxos is not guaranteed to reach the consensus maybe sometimes within a particular time bound

(Refer Slide Time: 30:36)



Paxos Algorithm

- Paxos has **rounds**; each round has a unique ballot id
- Rounds are asynchronous
 - Time synchronization not required
 - If you're in round j and hear a message from round $j+1$, abort everything and move over to round $j+1$
 - Use timeouts; may be pessimistic
- Each round itself broken into phases (which are also asynchronous)
 - Phase 1: A leader is elected (**Election**)
 - Phase 2: Leader proposes a value, processes ack
 - Phase 3: Leader multicasts final value

Cloud Computing and Distributed SystemsConsensus in Cloud Computing & Paxos

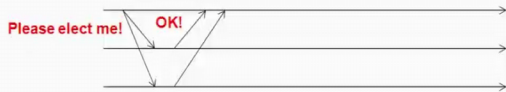
Let us see the Paxos algorithm. Paxos has divided the algorithm in and this operates in the rounds. So, the rounds are asynchronous, that is time synchronization is not required, but if you are in the round j and hear the messages from round j plus 1 then you abort everything and move over to the next to the round j plus 1. But that does not requires the time synchronization to do this. Timeouts can be used but maybe pessimistic so, each round itself broken into the phases which are also asynchronous.

So, let use see what are the phases this around of Paxos algorithm follows. So, the first phase is called the election. That is a leader is elected, second phase is called that the leader proposes a value. And also processes the acknowledgement ack. Phase 3 says that the leader multicast the final values.

(Refer Slide Time: 31:57)

Phase 1: Election

- Potential leader chooses a unique ballot id, higher than seen anything so far
- Sends to all processes
- Processes wait, respond once to highest ballot id
 - If potential leader sees a higher ballot id, it can't be a leader
 - Paxos tolerant to multiple leaders, but we'll only discuss 1 leader case
 - Processes also **log** received ballot ID on disk
- If a process has in a previous round decided on a value v' , it includes value v' in its response
- If **majority (i.e., quorum)** respond OK then you are the leader
 - If no one has majority, start new round
- (If things go right) A round cannot have two leaders (why?)



The diagram shows a potential leader on the left sending three arrows labeled 'Please elect me!' to three horizontal lines representing processes. The top two processes send arrows back labeled 'OK!'.

Cloud Computing and Distributed SystemsConsensus in Cloud Computing & Paxos

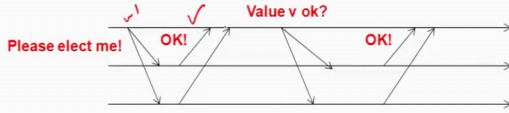
So, the potential leader chooses a unique ballot id; that means, the higher than anything seen so far is being chosen as the ballot ID. It sends it to all the processes. Processes wait and respond once to the highest seen ballot ID.

Now, if the potential leader sees a higher ballot ID then it cannot be the leader. Now the Paxos in some cases is tolerant to the multiple leaders, but let us without loss of generality let us discuss only one leader over here. So, the processes also log the received ballots on the disk. If a process has in the previous round decided a value v prime, it includes that value v prime in its response. If the majority that is the quorum responded then, that particular process will become the leader if no one has the majority then it will start again the new round. So, the things go right the round cannot have the 2 leaders.

(Refer Slide Time: 33:27)

Phase 2: Proposal

- Leader sends proposed value v to all
 - use $v=v'$ if some process already decided in a previous round and sent you its decided value v'
 - If multiple such v' received, use latest one
- Recipient logs on disk; responds OK



The diagram illustrates the Phase 2: Proposal process. It shows three horizontal lines representing processes. The top line is the leader. The leader sends a 'Please elect me!' message to the other two processes. Both respond with 'OK!'. The leader then sends a 'Value v ok?' message to both. Both respond with 'OK!'. The leader then sends a 'v!' message to both, which they receive.

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

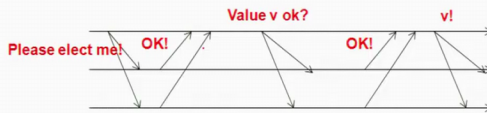
So, this is the phase one which is an election. The phase 2 which is called a proposal that is the leader sends the proposed value v to all, and v prime is the previous round value then the same value can be send. So, if multiple such v primes are received then in that case use the latest one.

Recipients logs all these values on the disk and responds to the OK messages here. So, they will send these values. And they will respond with the OK values.

(Refer Slide Time: 34:24)

Phase 3: Decision

- If leader hears a **majority** of OKs, it lets everyone know of the decision
- Recipients receive decision, log it on disk



The diagram illustrates the Phase 3: Decision process. It shows three horizontal lines representing processes. The top line is the leader. The leader sends a 'Please elect me!' message to the other two processes. Both respond with 'OK!'. The leader then sends a 'Value v ok?' message to both. Both respond with 'OK!'. The leader then sends a 'v!' message to both, which they receive.

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

Third phase is called decision if the leader hears a majority of ok. It let us everyone know of the decision. So, recipients receive the decision log on the disk.

(Refer Slide Time: 34:36)

Which is the point of No-Return?

- If/when a majority of processes hear proposed value and accept it (i.e., are about to/have respond(ed) with an OK!)
- Processes **may not know it yet**, but a decision has been made for the group
 - Even leader does not know it yet
- What if leader fails after that?
 - Keep having rounds until some round completes

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

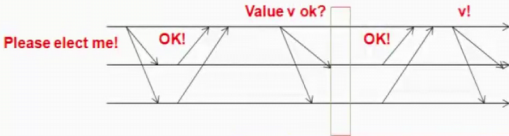
Now, at what stage the agreement is reached? That means, if the majority of the process hears the proposed value and accepted it; that means that is being checked with the help of this messages. Then it is known that the agreement is released and the processes may not know it yet, but the decision have been made in the rounds of this particular protocol Paxos.

But the decision has been made by the group even the leader does not know this particular decision at this point of time, before the algorithm completes. Now the question is if the leader fails, then what will happen? So, it will keep on iterating the rounds until some round completes it.

(Refer Slide Time: 35:54)

Safety

- If some round has a majority (i.e., quorum) hearing proposed value v' and accepting it, then subsequently at each round either: 1) the round chooses v' as decision or 2) the round fails
- **Proof:**
 - Potential leader waits for majority of OKs in Phase 1
 - At least one will contain v' (because two majorities or quorums always intersect)
 - It will choose to send out v' in Phase 2
- Success requires a majority, and any two majority sets intersect



Cloud Computing and Distributed SystemsConsensus in Cloud Computing & Paxos

So, let us see that this particular protocol Paxos achieves or guarantees the safety. Now if some round has the majority hearing the proposed value v prime and accepting it then subsequently in each round either the round chooses the same values v prime as the decision or the round fails.

The proof says that the potential leader wait for the majority of ok's in the phase 1. And at least one will contain v prime. And because if there are more than one majorities, but every message has to send only 1 ok. So, it will choose to send out v primes in the phase 2. Success requires a majority and any 2 majority sets the intersect.

(Refer Slide Time: 36:54)

What could go Wrong?

- **Process fails**
 - Majority does not include it
 - When process restarts, it uses log to retrieve a past decision (if any) and past-seen ballot ids. Tries to know of past decisions.
- **Leader fails**
 - Start another round
- **Messages dropped**
 - If too flaky, just start another round
- **Note that anyone can start a round any time**
- **Protocol may never end**
 - Impossibility result not violated
 - If things go well sometime in the future, consensus reached

Value v ok?

Please elect me! OK! OK! OK! v!

Cloud Computing and Distributed SystemsConsensus in Cloud Computing & Paxos

So, it will not validate the safety property; that means, even if the majority sets when we take the intersection. Even then we will found the values v prime to be reached in the agreement.

So, there while the rounds and the phases operates in the Paxos. There may be many type of crashes or a problems happens and let us see how tolerant it is in this particular scenarios. For example, a process fails. So, majority does not include it, and when the process restart it uses the log to retrieve the past decisions and past seen ballot ID's and tries to know the past decisions. Similarly, if the leader fails then it will start another round, messages are dropped and in that case you have to just start another round.

So, anyone can start around at any time and the protocol may never end; that is, impossibility results are not violated. So, if the things go well, sometime in the future consensus is reached, but that is not guaranteed.

(Refer Slide Time: 38:29)

What could go Wrong?

- A lot more!
- This is a highly simplified view of Paxos.
- **Lamport's original paper:**

The Part-Time Parliament

LESLIE LAMPORT
Digital Equipment Corporation

Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators. The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers. The Paxos parliament's protocol provides a new way of implementing the state-machine approach to the design of distributed systems.

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

So, here we will we can see here the original Lamports paper on Paxos; which is having a title that part time parliament can see and study conclusion.

(Refer Slide Time: 38:40)

Conclusion

- Consensus is a very important problem
 - **Equivalent to many important distributed computing problems that have to do with reliability**
- **Consensus is possible to solve in a synchronous system where message delays and processing delays are bounded**
- **Consensus is impossible to solve in an asynchronous system where these delays are unbounded**
- **Paxos protocol:** widely used implementation of a safe, eventually-live consensus protocol for asynchronous systems
 - Paxos (or variants) used in Apache Zookeeper, Google's Chubby system, Active Disk Paxos, and many other cloud computing systems

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

Consensus is very important problem. Equivalent to many important problems in the distributed systems, that have to deal with the reliability. Consensus is possible to be solved in a synchronous system where the message delays and the processing delays are bounded. Consensus problem is impossible to be solved in a model that is asynchronous model where these delays that is the messages and the processing are unbounded.

So, the Paxos is widely used protocol which will implement the safe and eventual live consensus protocol for asynchronous system, and is heavily used in very many industries systems by Yahoo; such as Apache Zookeeper and Google, Google's internal stack such that Chubby system and so on.

Thank you.