A Progressive Fault Detection and Service Recovery Mechanism in Mobile Agent Systems

Wong Tsz Yeung

Aug 26, 2002

Outline

- Introduction of the problem
- How to Solve the Problem
  - Server failure detection and recovery
  - Agent failure detection and recovery
  - Link failure
- Failure Detection and Recovery Mechanism Analysis
  - Liveness proof
  - Mechanism simplification analysis

Outline

- Reliability Evaluations
  - Using agent implementation
  - Using Stochastic Petri Net Simulation

Introduction of the problem

- Focus on designing a fault-tolerant mobile agent system
- The challenge is:
  - Guarantee the availability of the servers.
  - Guarantee the availability of the agents.
  - Preserve data consistency in both agents and servers.
  - Preserve the exactly-once property.
  - Guarantee the agent can eventually finish its tasks.

Introduction of the problem

- Fault-tolerance is classified into levels
  - Level 0: No tolerance to faults
  - Level 1: Server failure detection and recovery
  - Level 2: Agent failure detection and recovery
  - Level 3: Link failure

Level 0

- No tolerance to faults
  - When agent dies
    - because of server failure
    - because of faults inside agent
  - Application has to restart manually.
  - Affected server may leave an inconsistent state after recovery.

Level 1

- Server failure detection and recovery
  - Have a failure detection program running.
  - When a server restarts, abort all uncommitted transactions in the server.
    - This preserves data consistency
  - When the agent re-executes after the initial states
    - visited servers will be visited again
    - Violates exactly-once execution property

Level 2

- Agent failure detection and recovery
  - When server fails, agents resides are lost.
    - We aims to recover such loss in this level
  - By using checkpointing
    - We checkpoint agent internal data
    - We use checkpointed data to recover lost agents.

Level 2

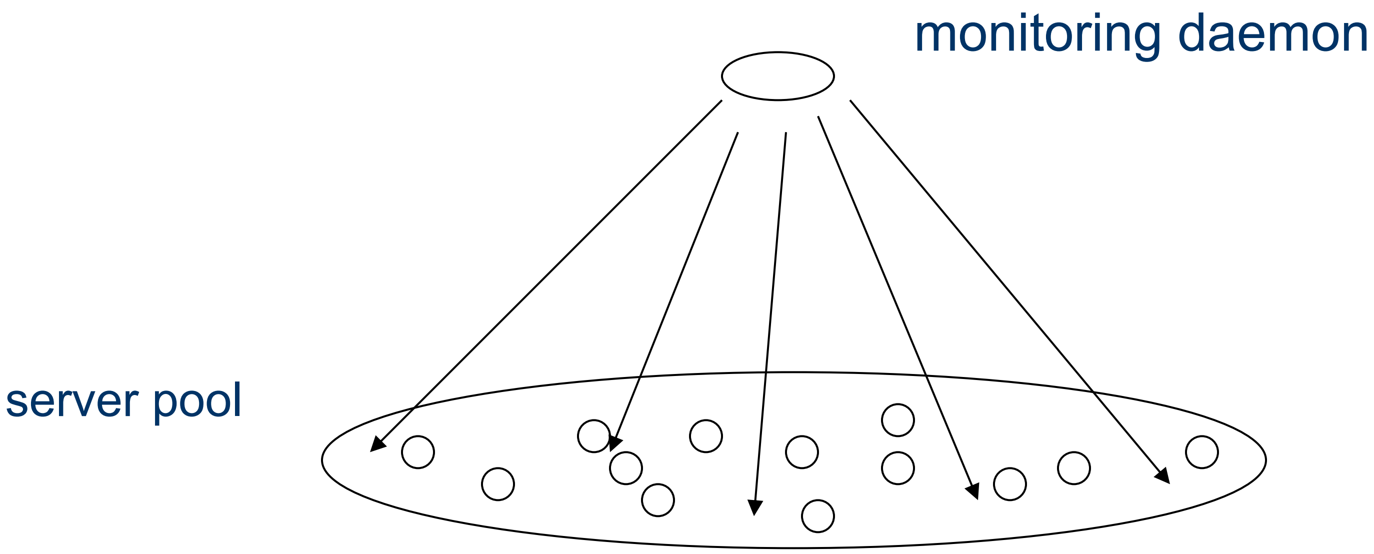
- Since we use checkpointed agent data
  - Agent data consistency is preserved
- Recovery of agent happens on the failed server
  - This preserves the exactly-once execution property.

Level 3

- Link Failure
  - We assume the agent agent is now ready to migrate from server *u* to server *v*, but a link failure happens
  - 3 scenarios
    - before the agent leaves *u*.
    - while the agent is traveling to *v*.
    - after the agent has reached *v*.
  - Different scenarios has different problems and corresponding solutions.

Design of Level 1 FT

- We have a global daemon which monitors all the servers.
- Single point of failure problem



Design of Level 1 FT

- When the daemon recovers a server
  - it aborts all the uncommitted transactions performed by those lost agents.
  - This preserves data consistency in the server.
- This technique is
  - easy to implement
  - can be deployed on every existing mobile agent platform, without modifying the platform.

Design of Level 2 FT

- We use cooperative agents.
  - Actual agent
  - Witness agent
- Actual agent performs actual computation for the user.
- Witness agent monitors the availability of actual agent.
  - It lags behind the actual agent.

Design of Level 2 FT

- In our protocol, actual agents are able to communicate with the witness agent
  - the message is not a broadcast one, but a peer-to-peer one
  - Actual agent can assume that the witness agent is in the previous server
  - Actual agent must know the address of the previous server

Protocol of Level 2 FT

arrive

leave

Arrive at *i*

Leave *i*

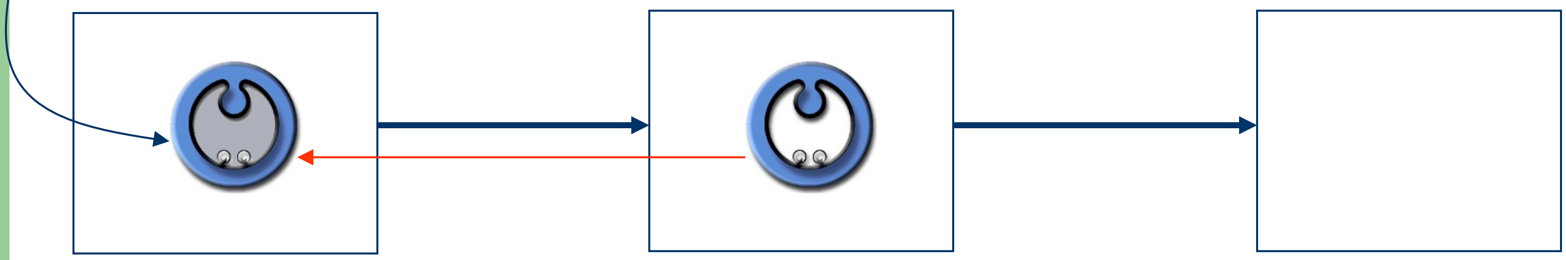
Agent messages box

Checkpointing happens!!

Server *i-1*

Server *i*

Server *i+1*



Arrive at *i*

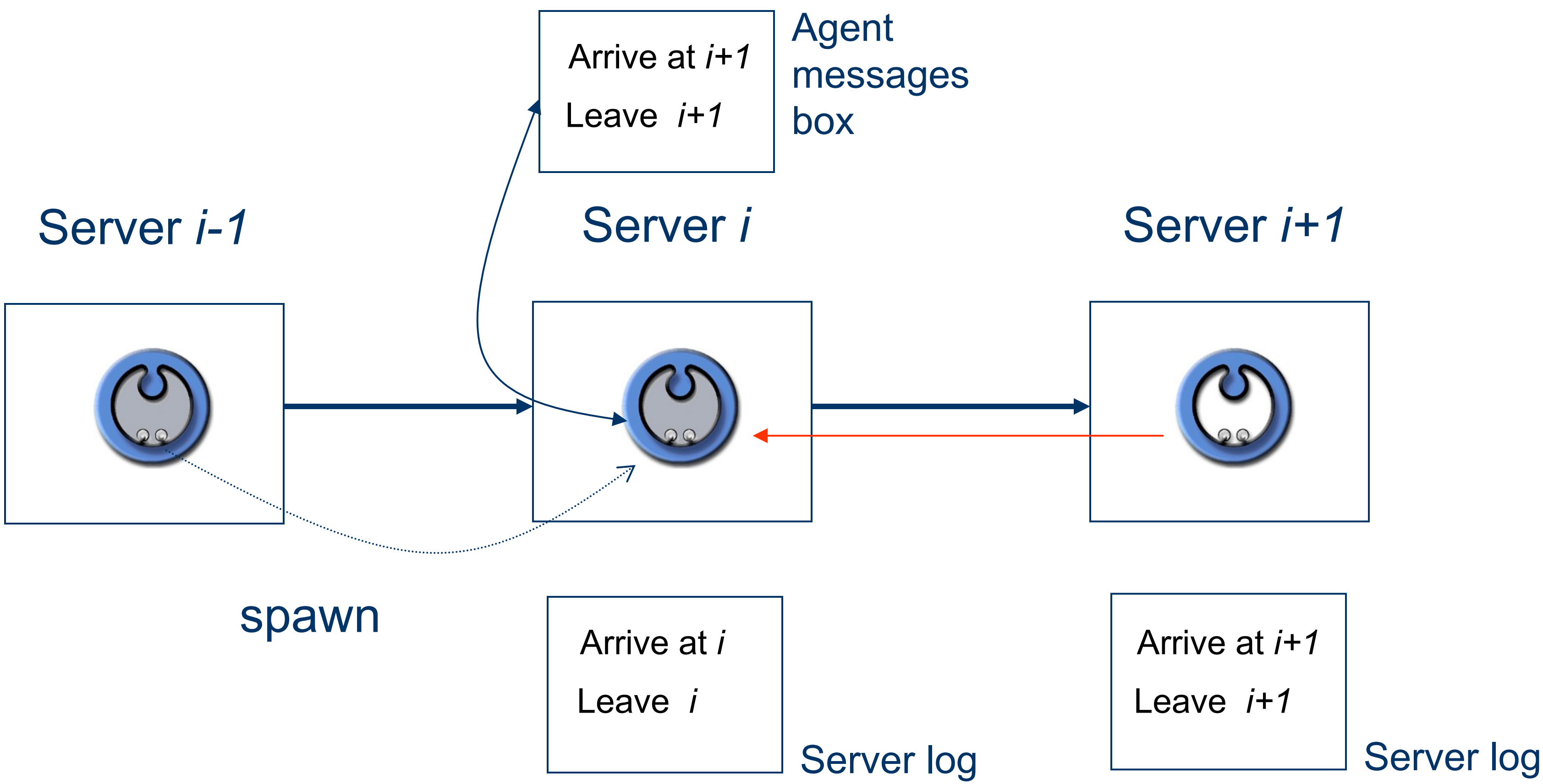
Leave *i*

Server log

Server log

Server log

Protocol of Level 2 FT



Failure and Recovery Scenarios

- We cover only cover stopping failures. (Byzantine failures do not exist)

- We handle most kinds of failures:
  - Witness agent fails to receive “arrive at i” message
  - Witness agent fails to receive “leave i” message
  - Witness agent failures

Missing arrive message

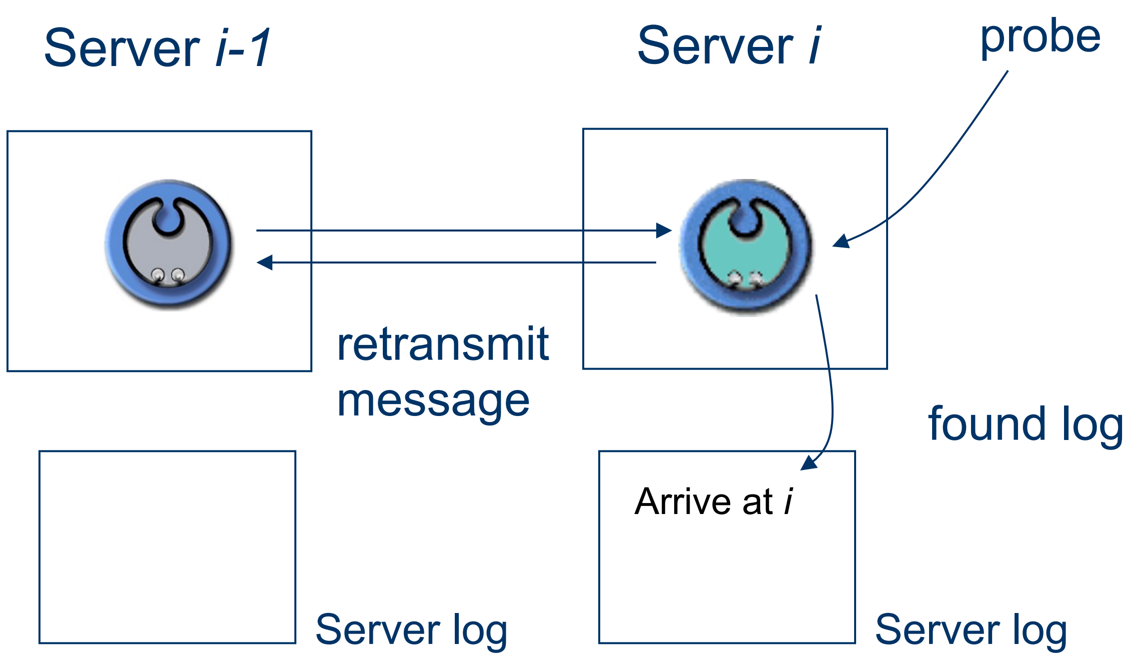
- The reason may be:

1. message is lost
2. message arrives after timeout period
3. actual agent dies when it is ready to leave server i-1
4. actual agent dies when it has just arrive at server i, without logging.
5. actual agent dies when it has just arrive at server i, with logging.

Next

Missing arrive message

- It is simple for the 1st and 2nd case.

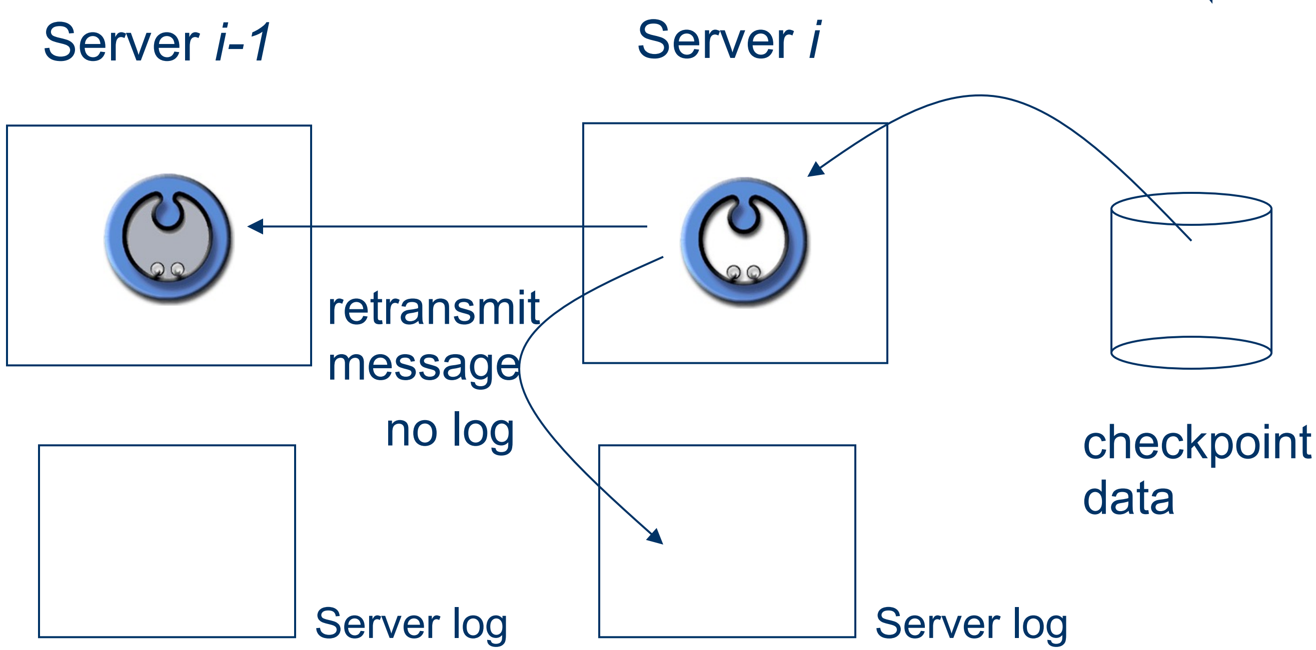


Back

Missing arrive message

- For the 3rd and 4th cases, recovery takes place.

Back



Missing arrive message

- For the 5th case, it results in missing detection.
  - since log appears in the server
  - the consequence is that “leave i” message never arrives.

Back

Missing leave message

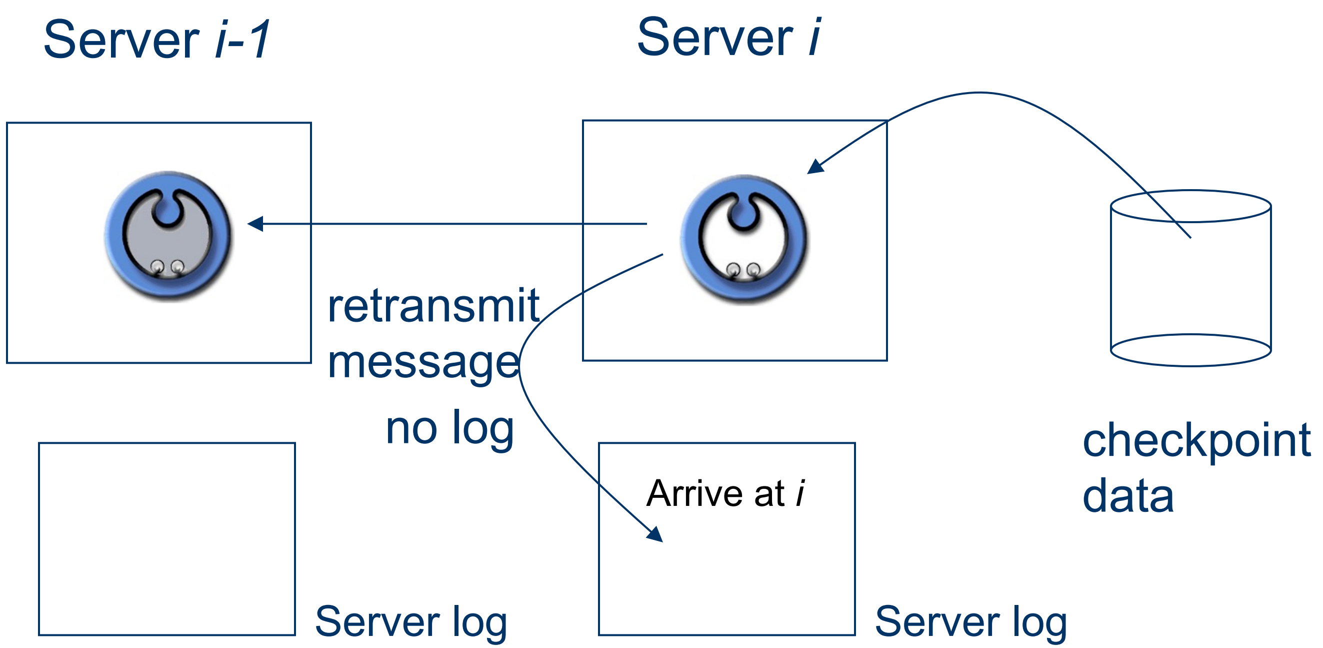
- The reason may be:

1. message is lost.
2. message arrives after timeout period
3. actual agent dies when it has just sent the “arrive at i” message
4. actual agent dies when it has just logged the message “leave i” message.

Next

Missing leave message

- The 3rd case is the same as the previous missing detection case.



Missing leave message

- In this case, the recovery action is the same as the previous section.
  - When failure happens, the agent should be performing computation.
  - So, when server recovers, the agent's computation has aborted.

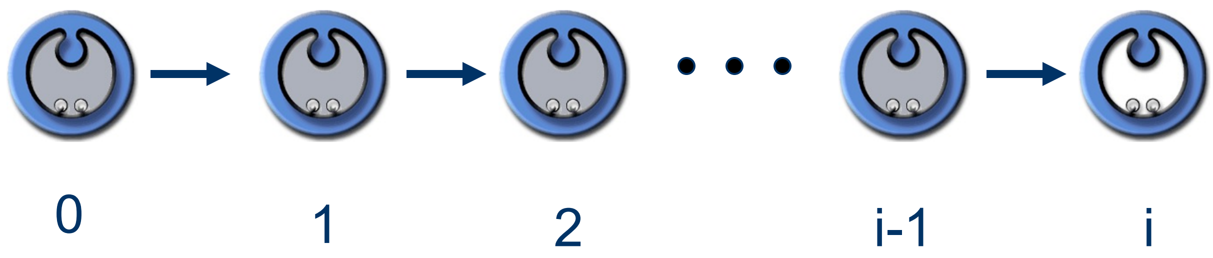
Back

Missing leave message

- This results in missing detection again.
  - This can be compensated by the 3rd case in the previous discussion.
  - It is because the witness will never receive “arrive i+1”.

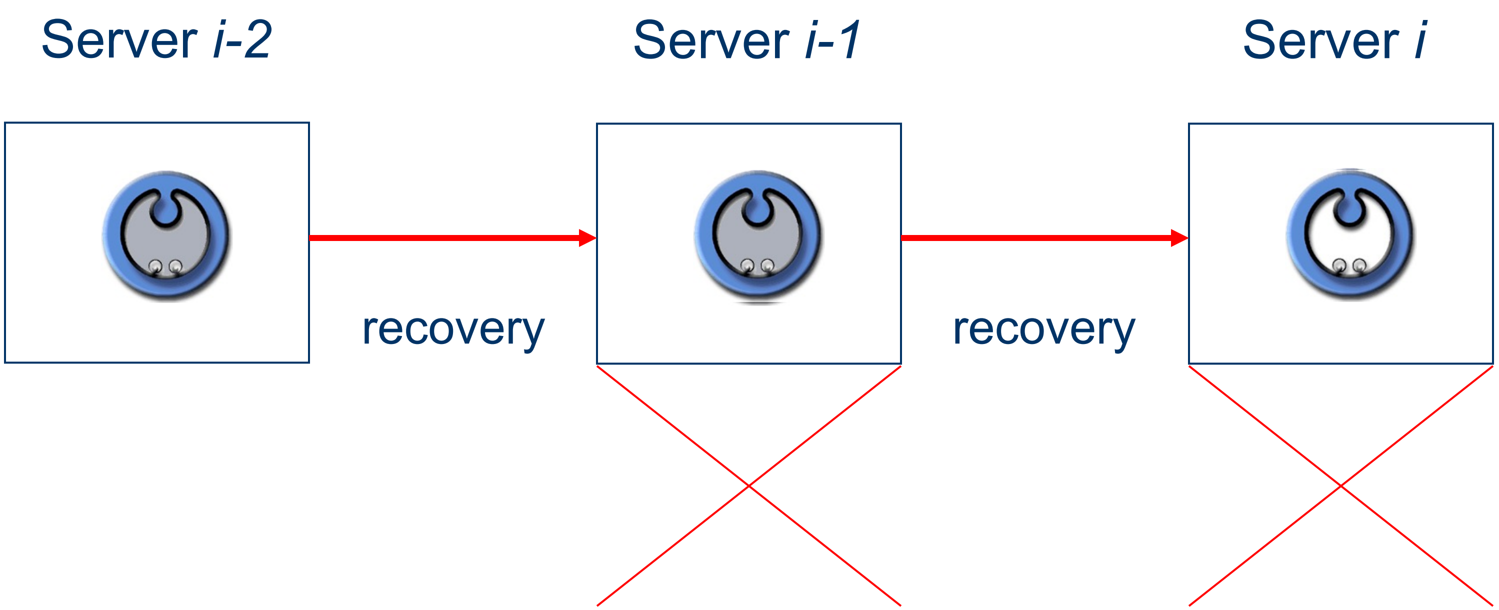
Witness Failure Scenarios

- There is a chain of witness agents leaves on the itinerary of the agent
  - The latest witness monitors the actual agent.
  - Other witnesses monitor the witness that is before it.



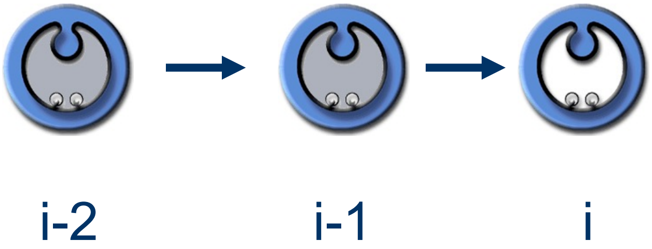
Witnessing dependency

Witness Failure Scenarios



Simplification

- Assume that 2-server failure would not happen
  - We can simplify our witnessing dependency



Simplification

- If failure strikes server i-1
  - witness on server i-2 can recover witness on server i-1
- If failure strikes server i-2
  - Will not recover it
  - Because within a short period, no failure would happen

Analysis – Liveness proof

- Notations
  - We define several timeouts
    - T_recover: The timeout of waiting for a server to be recovered.
    - T_arrive: the timeout of waiting for the *arrive message*.
    - T_leave: the timeout of waiting for the *leave message*.
    - T_alive: the timeout of waiting for the *alive message*.

  - Also, define several constants
    - *r_s*: the maximum time for a server to be recovered when detected.
    - r_a: the maximum time for an actual agent to be recovered.
    - *a*: the maximum agent traveling time between 2 servers.
    - *m*: the maximum message traveling time between 2 servers.
    - *e*: the maximum execution time for an agent.

Analysis – Liveness proof

- If the system is blocked forever, one of the three timeouts will reach infinity.
- The outline of the proof:
  - derive the lower and the upper bounds of the timeouts
  - Given that the itinerary of the agent is of finite length and infinite number of failures, if none of the timeouts approach infinity, the system is blocking-free.

Analysis – Liveness proof

- Level 1 FT analysis
  - A failed server will eventually be recovered, the time bound is:

$$r_s \leq T_{recover} \leq nr_s$$

  - In the worse case, all servers are stopped.
  - Need to recover *n* servers.

Analysis – Liveness proof

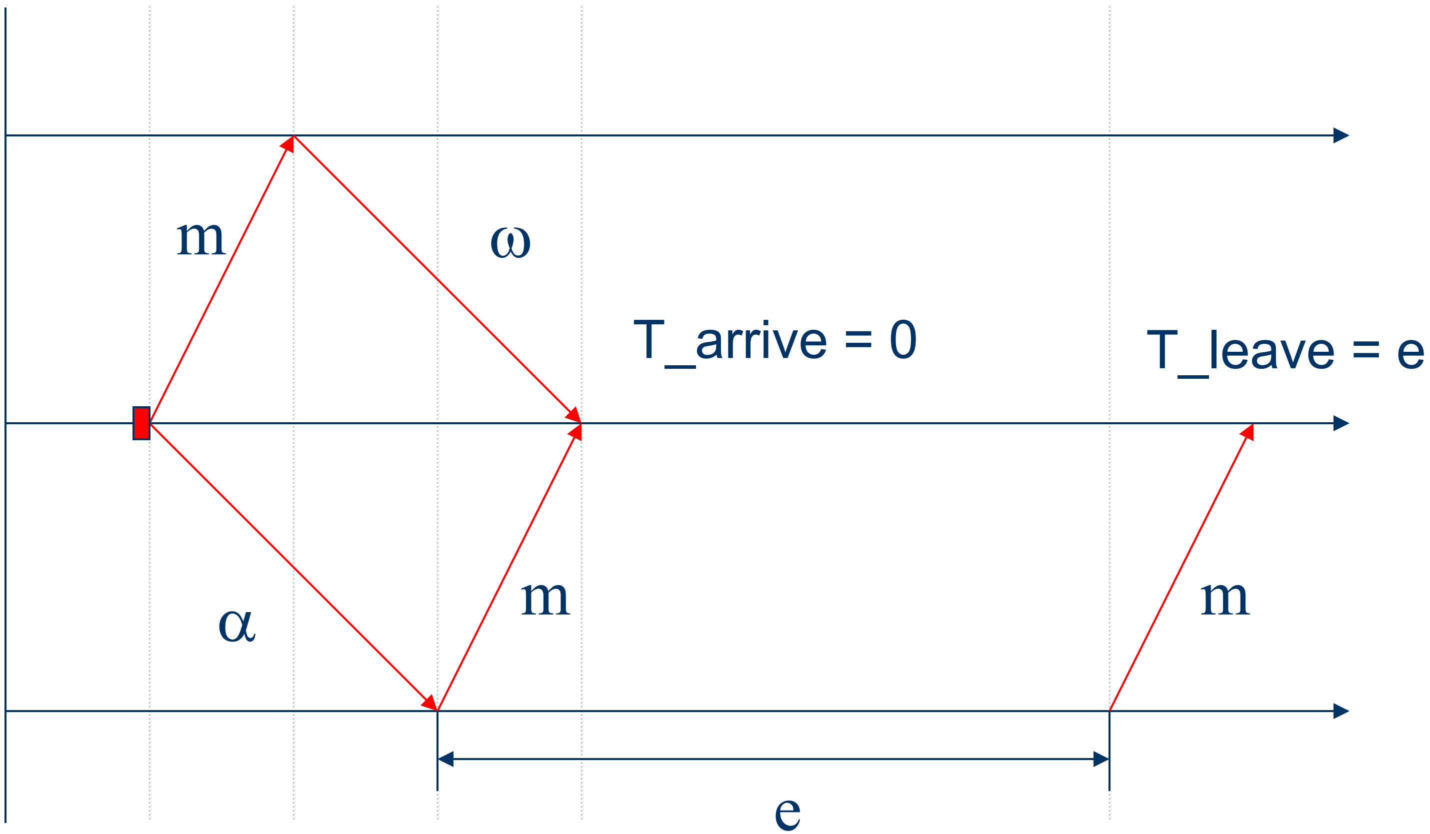
- Level 2 FT analysis
  - We derive the lower bounds for the timeouts

$$T_{arrive} \geq 0$$

$$T_{leave} \geq e$$

$$T_{alive} \geq a + m$$

Analysis – Liveness proof



Analysis – Liveness proof

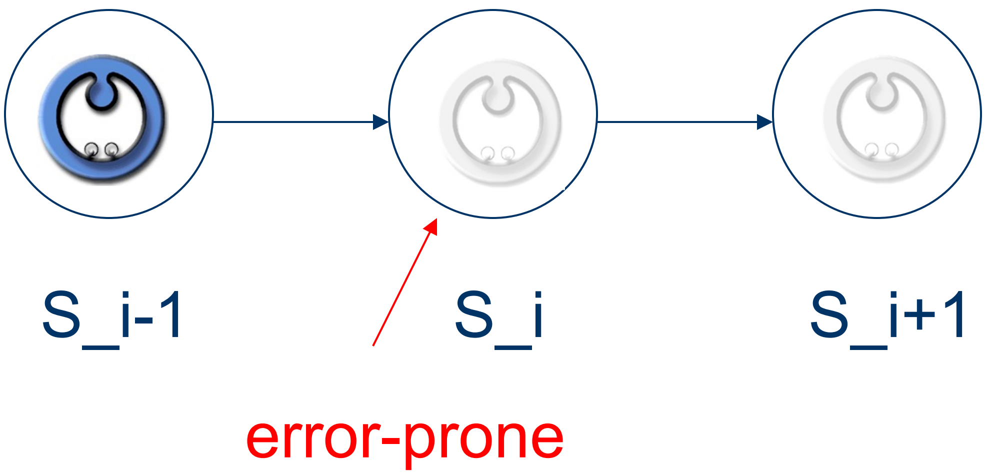
- We define the failure inter-arrival time be τ
- If the system is not blocked forever,

$$a + e \leq \tau \leq \infty$$

  - 2 cases are needed to be considered.
    - Does the actual agent have enough time to migrate from one host to another?
    - Also, does the witness agent have enough time to migrate?

Analysis- Liveness proof

- Assume that all failures are happening in S_i
  - During the actual agent is migrating, there should be no failures
  - So, the required time is <u>*a+e*</u>.

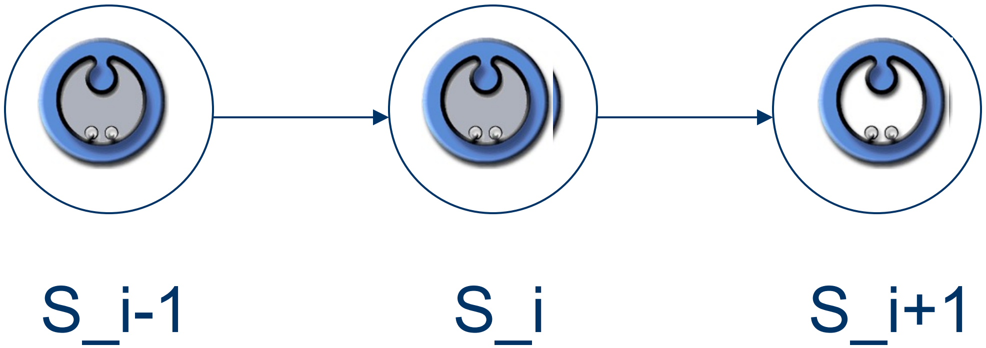


Analysis – Liveness proof

- Again, assume that all failures are happening in S_i
  - The required time

    = a + min(T_arrive) + min(T_leave)

    = a + e



Analysis – Liveness proof

- Useful results:

$$0 \le T_{arrive} \le T_{recover} + a + r_a$$

$$e \le T_{leave} \le k(T_{recover} + a + r_a) + (k-1)e + 2m$$

where $a + e \le \tau \le a + e + r_a$

where *k* is the number of failures

$$\max(a + m, e) \le T_{alive} \le T_{recover} + 2a + 2m$$

Analysis – Liveness proof

- By the above results, we conclude that:
  - The system is blocked iff all failures is happening on one server, and $a + e \le \tau \le a + e + r_a$
  - It follows from the upper and lower bounds of T_arrive, T_leave, and T_alive.

Simplification Analysis

- We define the following notation:
  - Define T be the inter-arrival time of the failures throughout the system.

$$\mathrm{T} > \max\begin{pmatrix} a+e \\ 2a+m \\ T_{recover} + a + m \end{pmatrix}$$

Link Failure

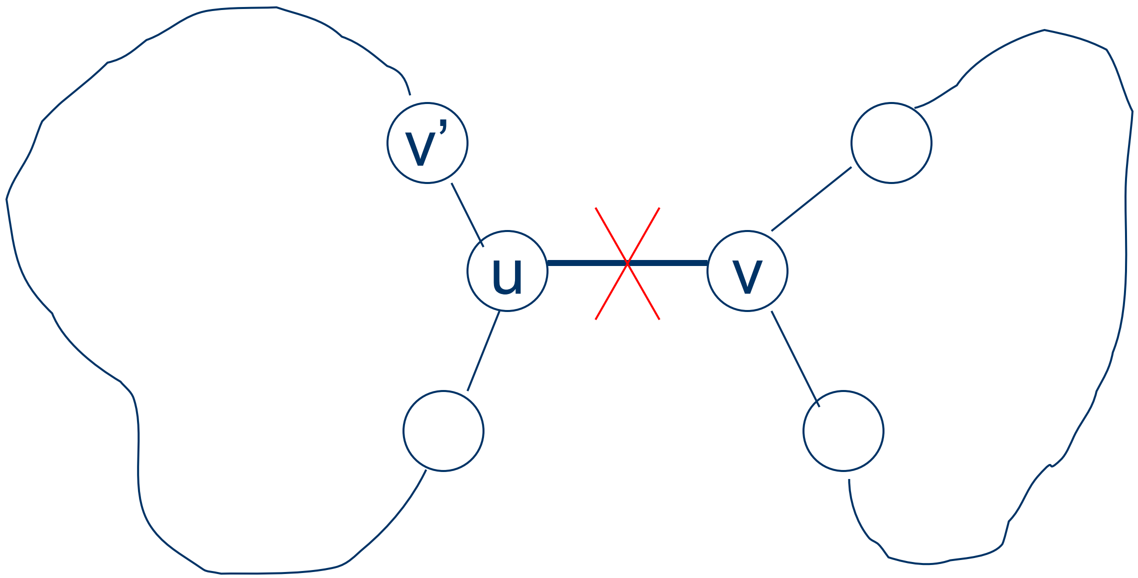
- Link failure is beyond the control of mobile agents system.
- Assume that the actual agent is ready to leave server *u* and migrate to *v*. Then, a link failure happens:
  - before the agent leaves *u*.
  - while the agent is traveling to *v*.
  - after the agent has reached *v*.
- We propose solutions to remedy these problems.

Link Failure

- Failure happens before the agent leaves *u*:
  - Problem:
    - the agent cannot proceed.
    - the agent waits in server *u* until the link is recovered.
  - Solution:
    - Travel to server *v'* instead of *v* based on number of migration trials.
    - Technical problem: Knowledge of the locations of the unvisited servers.

Link Failure

- If network partitioning happens:



Link Failure

- Failure happens while the agent is traveling to *v*:
  - Problem:
    - The agent is lost. Recovery is required.
    - However, the witness agent cannot proceed to server *v*.
  - Solution:
    - The witness agent cannot recovery the actual agent in another server, say *v'*.
    - Have to wait until the link is recovered.

Link Failure

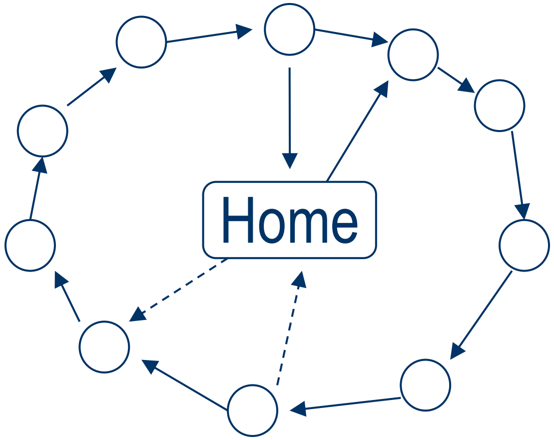
- Failure happens after the agent has arrived at *v*:
  - Problem:
    - The actual agent survives.
    - Messages between *u* and *v* cannot reach the destinations.
    - Witness agent cannot follow the actual agent.

Link Failure

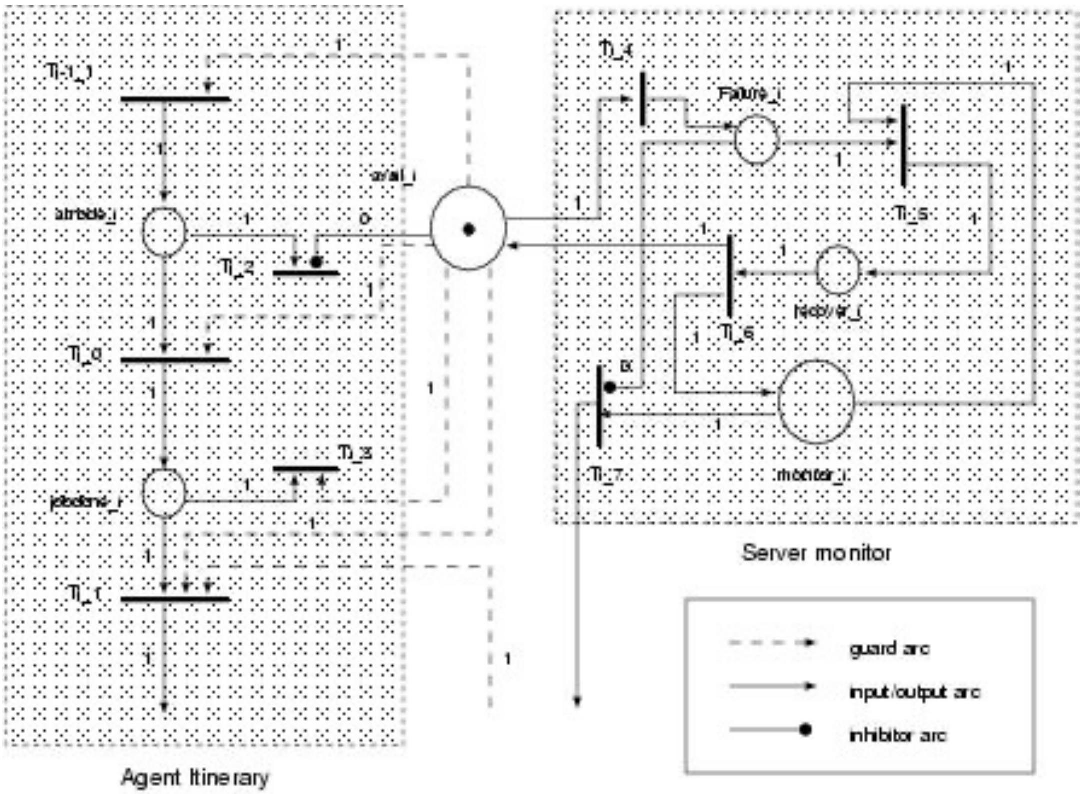
- Solution:
  - The actual agent keeps on advancing until:
    - It is lost in one of the servers.
      - After the link failure is recovered, the probe can eventually find such a failure.
    - It has reached the destination.
      - The probe can finally catch up.

Reliability Evaluation

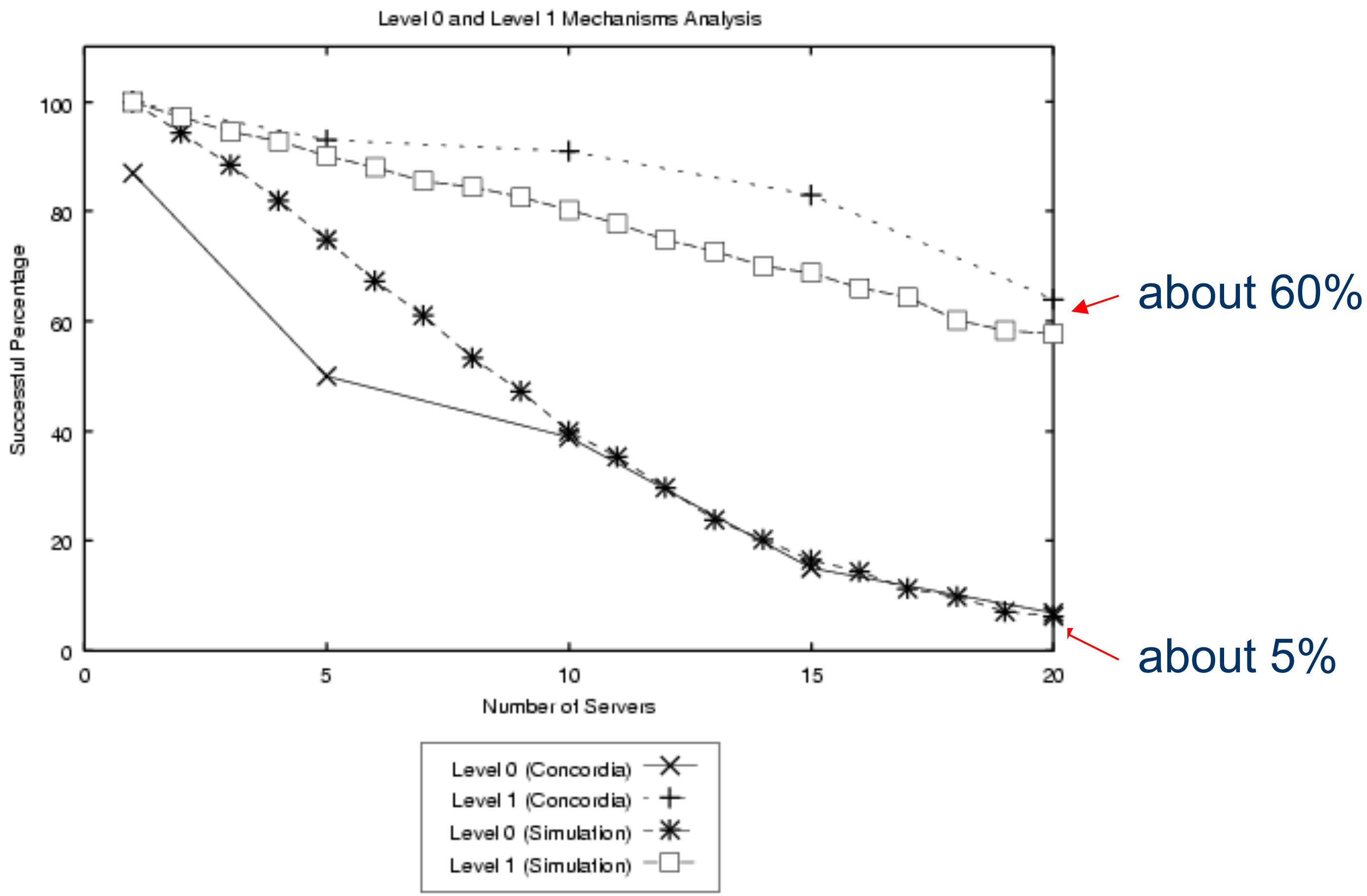
- The results are obtained by
  - an agent system implementation using Concordia.
  - simulation using Stochastic Petri Net.
  - aim: to measure the percentage of successful round-trip-travel.



Reliability Evaluation

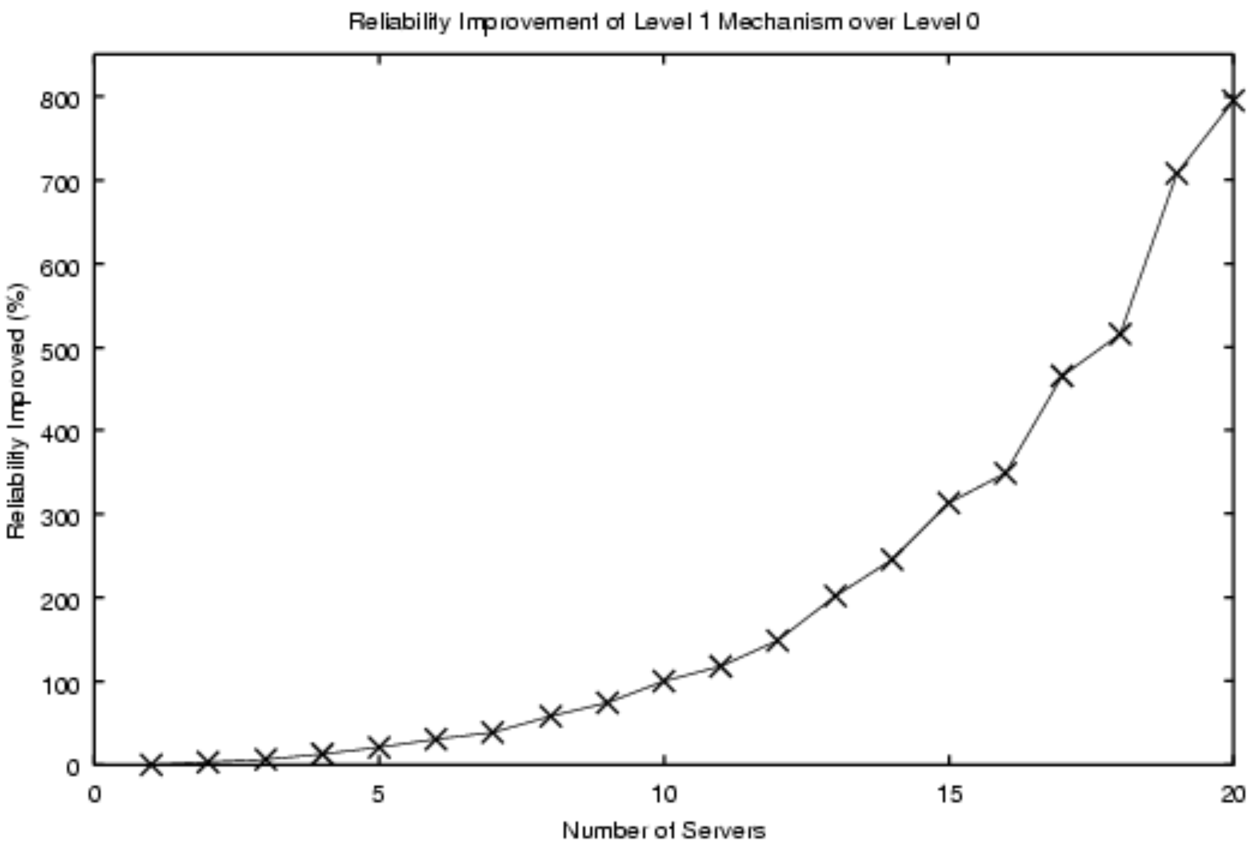


Reliability Evaluation



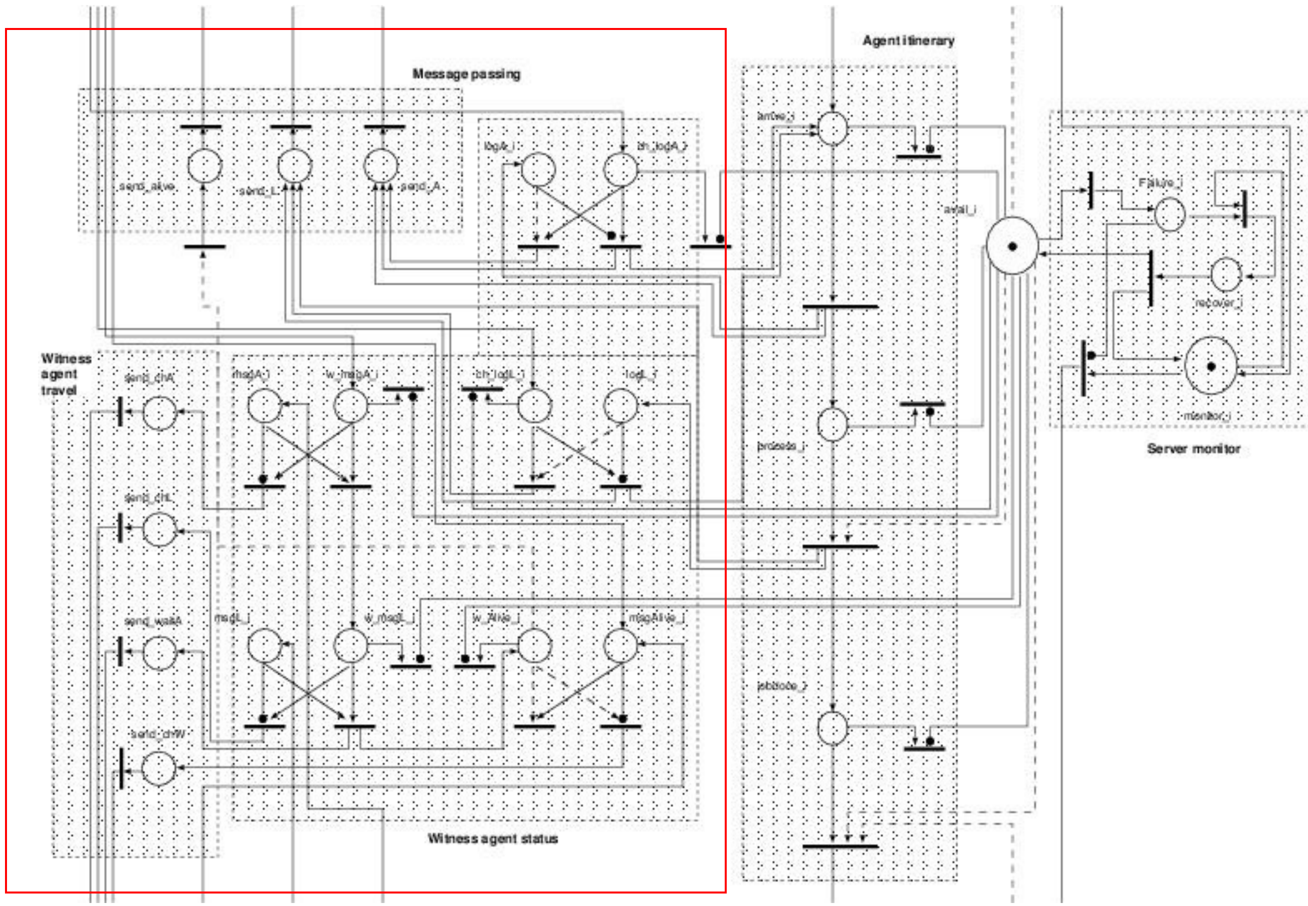
Reliability Evaluation

about 800%

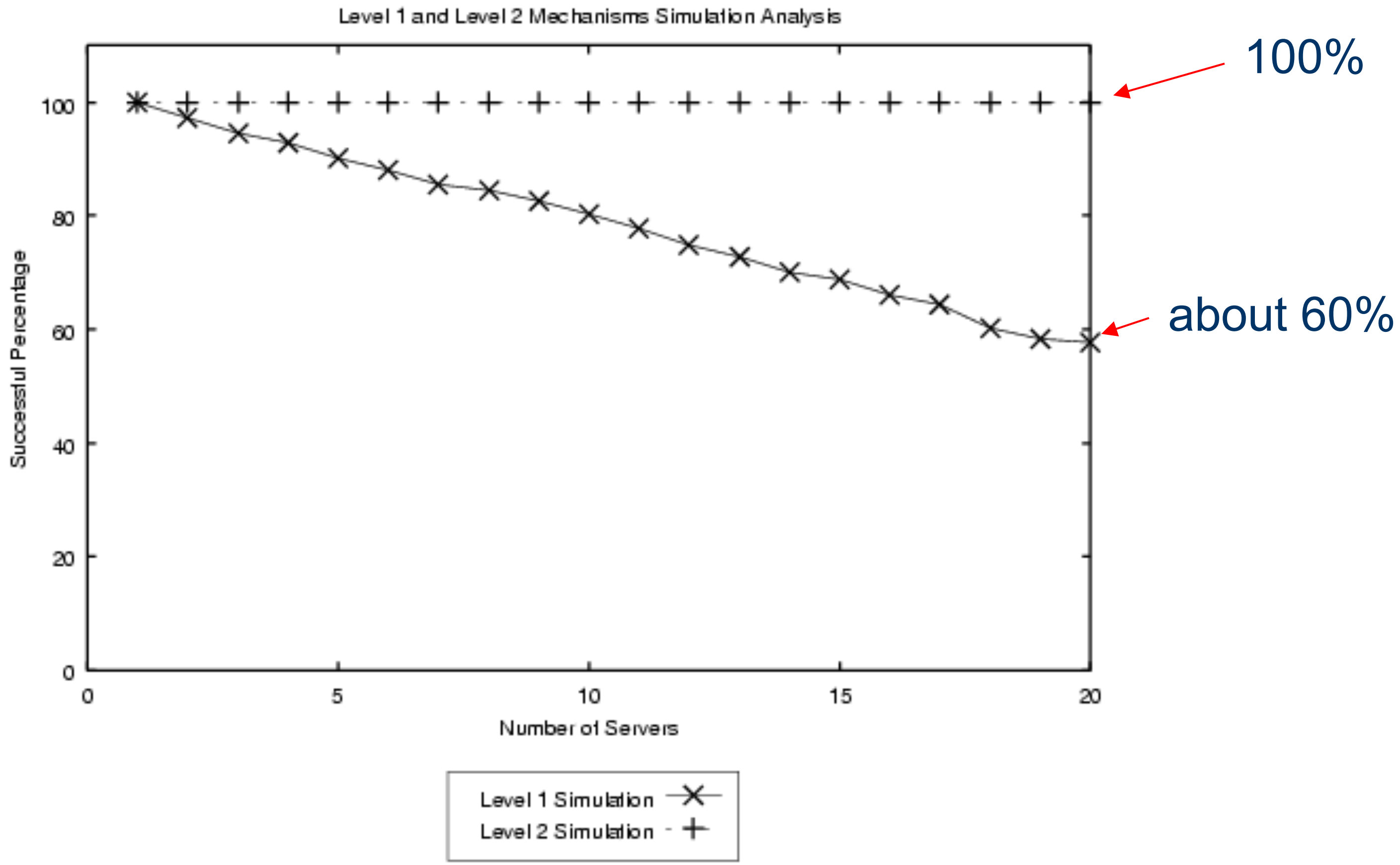


Reliability Evaluation

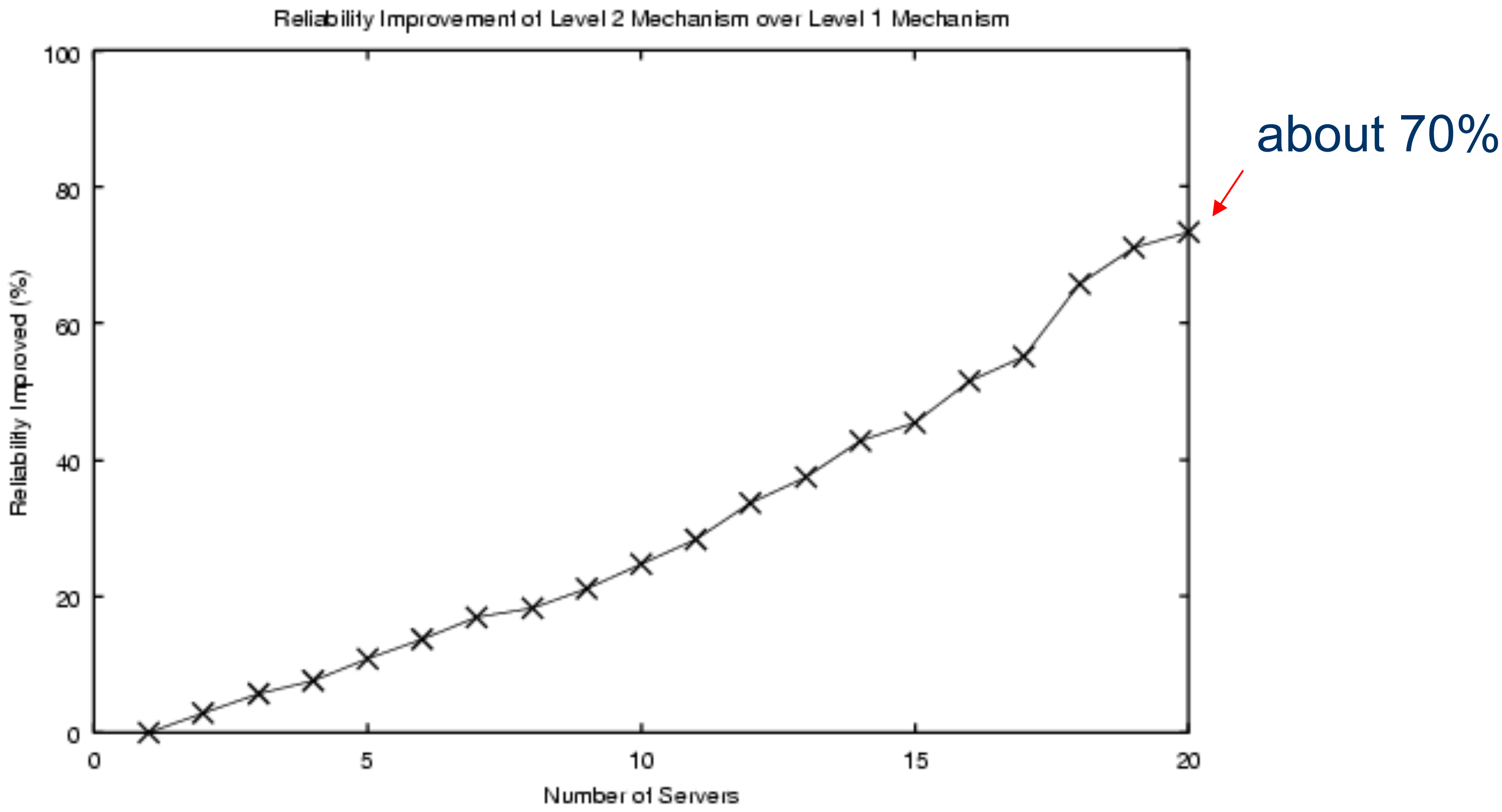
For agent failure detection only



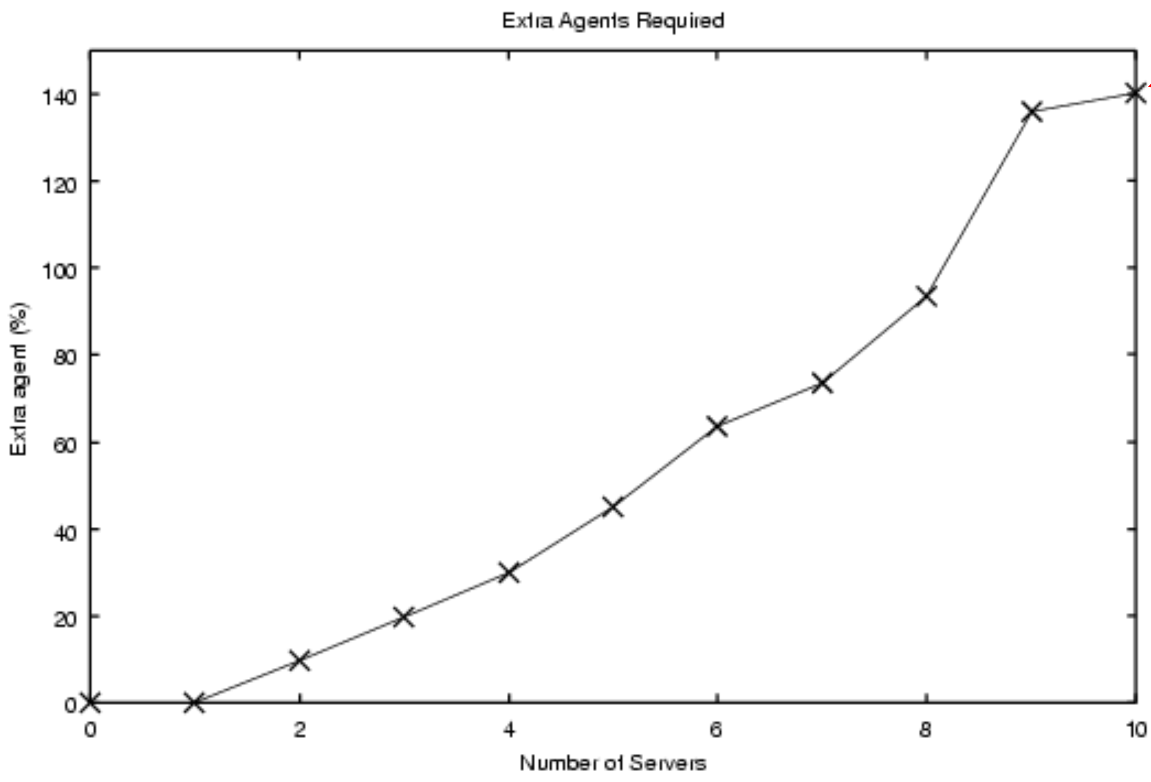
Reliability Evaluation



Reliability Evaluation



Reliability Evaluation



about 140%

$$\frac{\text{\# of recovered agent}}{\text{\# of original agent}}$$

Conclusion

- Categorize the fault-tolerance of mobile agent system.
- Designed a scheme for both server and agent failure detection and recovery.
- Analyzed most failure scenarios in mobile agent systems.
- Conducted performance evaluations which show
  - Our scheme is a promising technique
  - Trade-off between cost and levels of reliability

Future Work

- Model and perform simulations on Level 3 fault-tolerant mechanism.
- More detailed analysis is required.
- Extended stopping failures to Byzantine failures.

THE END

Q & A Session