# CHALMERS
## EXAMINATION / TENTAMEN

| Course code/ kurskod | Course name / kursnamn | | | |
|---|---|---|---|---|
| TDA 297 | Distributed System, Advanced | | | |
| Anonymous code<br>Anonym kod | | Examination date<br>Tentamensdatum | Number of pages<br>Antal blad | Grade<br>Betyg |
| TDA297 –10 | | 2014-03-12 | 15 | |

| Solved task<br>Behandlade uppgifter.<br>No / nr | | Points per task<br>Poäng på uppgiften. | Observe: Areas with bold contour are to be completed by the teacher.<br><br>Anmärkning: Rutor inom bred kontur ifylles av lärare. |
|---|---|---|---|
| 1 | X | 9 | |
| 2 | X | 7 | |
| 3 | X | 3 | |
| 4 | X | 9 | |
| 5 | X | 14 | |
| 6 | X | 10 | |
| 7 | | | |
| 8 | | 4 | |
| 9 | | | |
| 10 | | | |
| 11 | | | |
| 12 | | | |
| 13 | | | |
| 14 | | | |
| 15 | | | |
| 16 | | | |
| 17 | | | |
| 18 | | | |
| Total examination points<br>Summa poäng på tentamen | | 56 | |

[redacted]

3

| CHALMERS | Anonymous code / Anonym kod: TDA 297-10 | Points for question (to be filled in by teacher) / Poäng på uppgiften (ifylles av lärare) | Consecutive page no. / Löpande sid nr: 1 <br> Question no. / Uppgift nr: 1 |
|---|---|---|---|

1.

a) Linearizability:

In a concurrent execution, there exists a sequential execution that contains the same operations, is legal (obeys the rules of ADT) and preserves the real-time order of all the operations

(4)

Sequential Consistency:

In a concurrent execution, there exists a sequential execution that contains the same operations, is legal and it preserves the order of operations from the same ~~sender~~ process

b) Sequential Consistency is NOT composable.

example:

| | O1 | O2 | O3 |
|---|---|---|---|
| processA: | enq(Q1,X) | enq(Q2,Y | deq(Q2,Y) |
| process B: | enq(Q2,X) | enq(Q1, Y | deq(Q1, X) |
| | O11 | O12 | O13 |

suppose there are processes A and B, A executes O1, O2, O3; B executes O11, O12, O13.

From each queue's perspective, the execution is sequentil consistent because ther order of each sender is preserved

| | | | |
|---|---|---|---|
| Q1: | enq(Q1,X) | enq (Q1,Y) | deq (Q1, X) |
| | O1 | O12 | O13 |
| Q2: | enq(Q2,Y) | enq (Q2,X) | deq(Q2,Y) |
| | O2 | O11 | O3 |

→ Con't.

3

CHALMERS Anonymous code / Anonym kod: TDA 297-10

Points for question / Poäng pa uppgiften

Consecutive page no. / Löpande sid nr: 2

Question no. / Uppgift nr: 1

but the ~~ex~~ecutions on the two queue cannot be combined and still results in sequential consistency because deq($Q_1$, x) requires enq($Q_1$, x) happens before enq($Q_1$, Y) and deq($Q_2$ Y) requires enq($Q_2$, Y) happens before enq($Q_2$, x).

i.e., $O_1$ happens before $O_{12}$ and $O_2$ happens before $O_{11}$, which is impossible to achieve. The order of operations from the same sender cannot be preserved.

So Sequential Consistency is not composable.

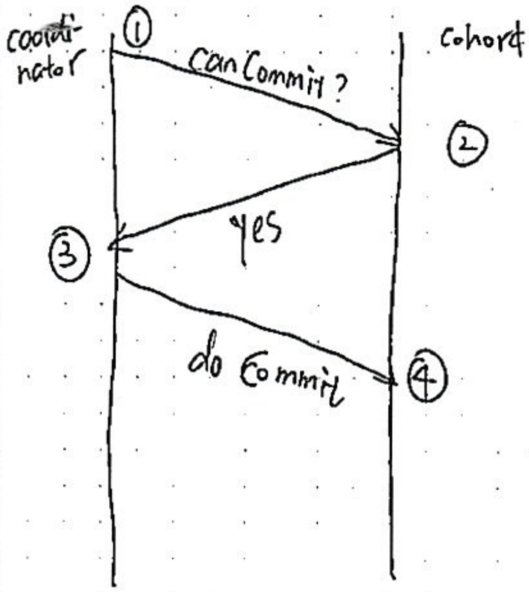
(5)

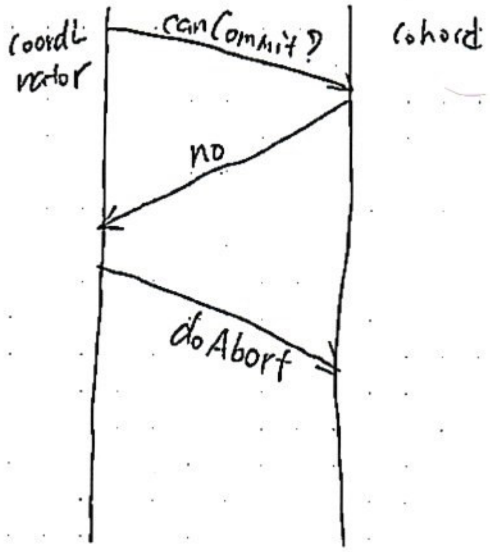
Answer only one question on this page. Do not write on the back of this paper
Behandla endast en uppgift pa detta blad. Skriv ej pa baksidan

3

CHALMERS | Anonymous code / Anonym kod: TDA 297-10 | Points for question (to be filled in by teacher) / Poäng pa uppgiften (fylles av läraren) | Consecutive page no. / Löpande sid nr: 3 | Question no. / Uppgift nr: 2

2. Two phase commit.

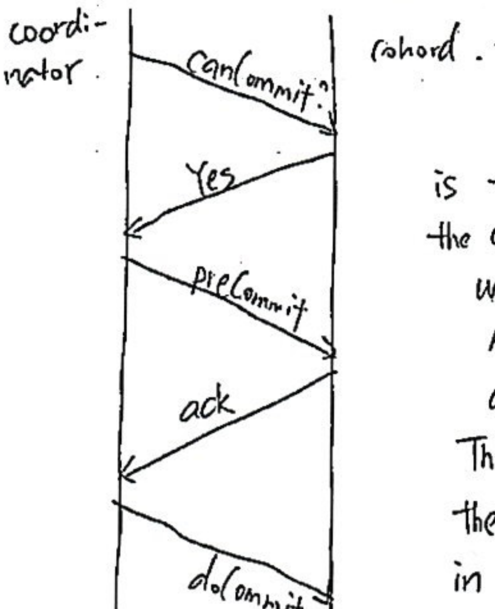



In the two phase commit protocal, first phase is the voting phase. Coordinator sends canCommit to all participant. If every body replies yes, the coordinator replies doCommit and every participant executes commit. (the second phase)

If any participant aborts, the coordinator sends do abort to everyone and all participaints abort.

Three phase commit




The difference of three-phase commit is that after all participants vote for yes, the coordinctor first sends a prepare commit which the participant will ack.

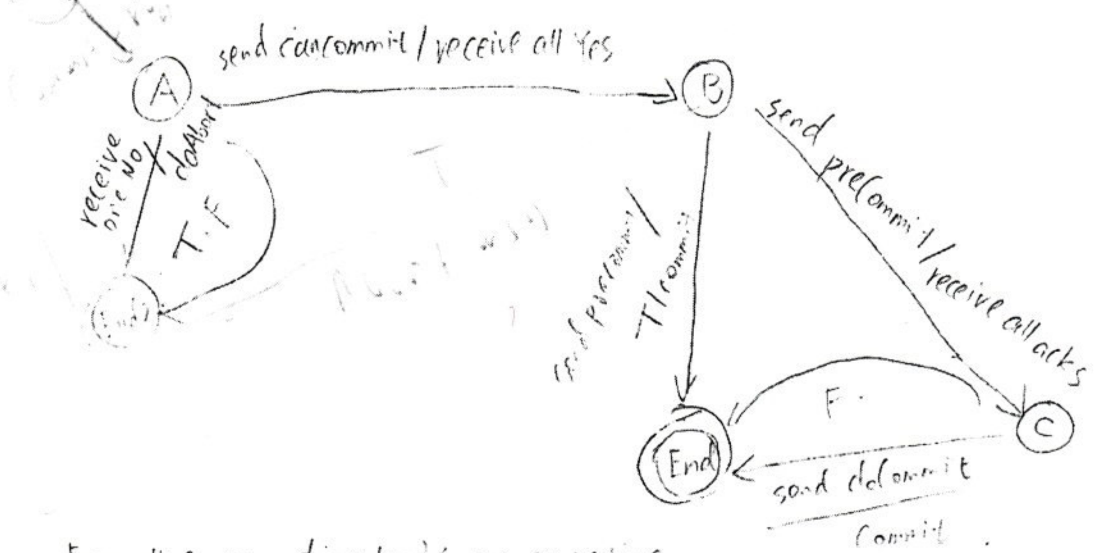
A do Commit is only sent when all the acks are received by the coordinator

The three-phase commit protocal solves the blocking issue that might happen in the two-phase commit. If the coordinator crashed in ② → ③, the participant who voted Yes will block to wait for the decision. It cannot safely abort because there might be participant who already completes the commit → Con't.

TDA 297-10

By adding a pre-commit phase in the 3-phase protocol, before the pre-commit commit messge is received, All the participants can still safely abort in case of timeout. Even if the coordinator crashes right before sending doCommit, the participants can continue commiting as the decision is saved in a permanent memory.

The state machine is as the following: (timeout = T
failure : F
for the coordinator




From the coordinator's perspective.

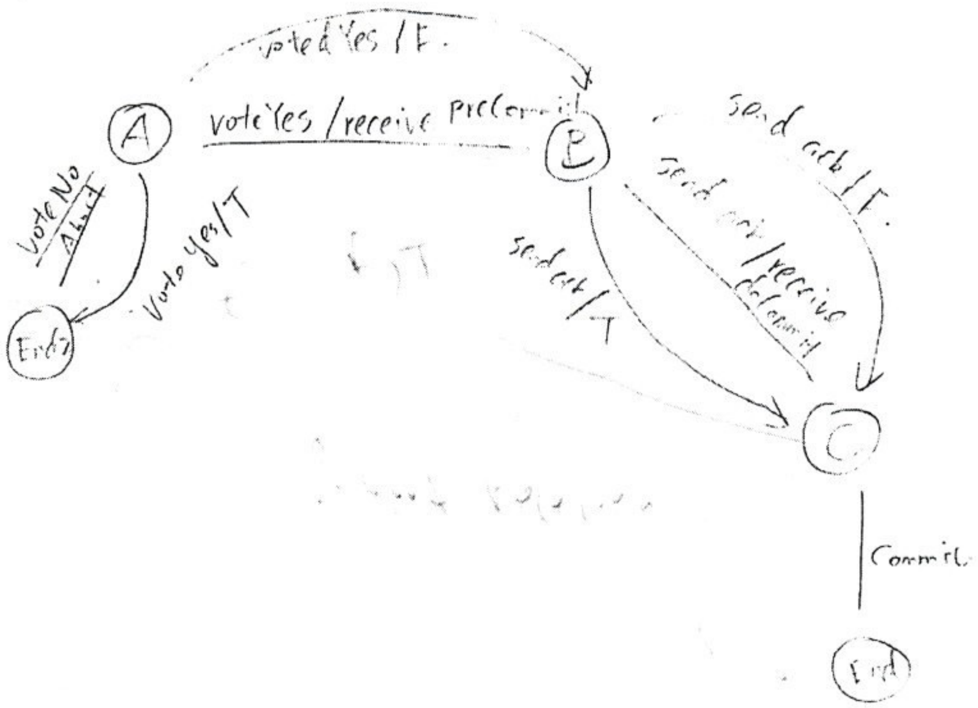
Ⓐ → Ⓑ → Ⓒ → (End) is the complete perfect sceinario of a successful commit. Ⓐ → (End) is when one process abort.

If timeout is detected before the precommit phase Ⓐ → (End), coordinator chooses to abort. Otherwise coordinator chooses to commit Ⓑ → (End) because the crashed participant that causes the timeout has voted yes and this decision will be recovered from the permanent memory.

Similarly, if the coordinator crashes before Ⓑ and then recovers, it should abort. After Ⓑ, just continue from where it left off.

TDA 297-10

for the cohord.




from a participant's perspective.

(A) - (B) - (C) - (End) is the full scinario of a successful commit

If a participant votes yes and then detects a timeout of the coordinator, it chooses to abort (A) -> (End)
if the timeout is detected after sending ack, (B) -> (C)
the participant chooses to commit as it is certain that all other participants have voted yes

If a participant crashes and recovers after it has voted, it should proceed from where it left off. If the vote ~~was~~ is yes before the process crashes, the updates to be commited is saved in the permanent memory.

6

3

TDA 297 -10

3. Causal broadcast :

For broadcasted message m1 and m2, if m1 → m2 , all the processes should deliver m1 before they deliver m2

The causal broadcast property requires that ALL the processes should deliver m1 before m2 if m1 → m2. As long as any single process delivers m2 before m1 in this case, the causal broadcast fails, the property is broken

However, the property mentioned in the question only looks at one process. This property has a much smaller scope than the causal broadcast property. It does not care if every process must preserve the same delivery order

The given property is a subset and it should hold for every $m_1$ such that $m_1 \rightarrow m_2$

(3)

3

CHALMERS Anonymous code / Anonym kod: TDA 297-10

Points for question / Poäng pa uppgiften

Consecutive page no. / Löpande sid nr: 7

Question no. / Uppgift nr: 4

4 a) Pi initiates the election in a ring of P1, P2, P3 .... Pn
leader Election_init :

proposal$_i$ := {ID$_{pi}$};
forward proposal$_i$ to the next neighbour;

Upon receiving proposal$_i$ at Pj :
if (ID$_{pj}$ ≠ i)
proposal$_i$ = proposal$_i$ ∪ {ID$_{pj}$}
forward proposal$_i$ to the next neighbour;
// check if not the initiator, append its own id in the proposal set and foward the msg
endif;

if (ID$_{pj}$ == i)
Leader = Max (proposal$_i$)
end if

(4)

// if a process receives a proposal set that is initiated by himself, then it means he has collected te IDs of all the processes in the ring and the leader can be chosen. (example: the leader withe the max id will be the leader)

b) Time complexity: $O(n^2)$ ×
Communication complexity: $O(n^2)$ ✓ (2)

The worst case is when the IDs are randomized and all the processes initiated the election one round after another

3

| CHALMERS | Anonymous code / Anonym kod | Points for question / Poäng pa uppgiften | Consecutive page no. / Löpande sid nr |
|---|---|---|---|
| | TDA 297-10 | | 8 |
| | | | Question no. / Uppgift nr: 4 |

c) It is NOT possible to design a symmetric algorithm for leader election.

Proof: proof towards contradiction

∃A that is symmetric and chooses a leader
all the processes are identical and
they must take the same steps and
always end up in the same status

Suppose the leader is elected in the ith step
After all the processes finish executing step i,
there will be one process Pk that is entitled
(3) the leader and it should be in a different
status than all other processors.

It is a contradiction because all
the processes are identical so Pk will
not be differentiated

Thus, it is Not possible to find such
an algorithm.

TDA297-10 5 9

5 a) Three phase commit is not suited for availability because it ensures very strict consistency that all the replicas must be identical. In case of one server fails, this server will choose to abort. The client request will thus be rejected as the client fails to collect all the YES votes.

(5) depends where it fails

The Replication system is sequential consistent and is linearizable. ✓

One property of the system is "the client performs only one storage operation at a time, waiting for each operation to complete before starting the next one". Under 3-phase commit protocol, each completion of an operation is a result of the two servers reaching an agreement, performing identical executions. Thus both servers will preserve the same and the real-time ordering of all the operations, which meet the criteria of linearizability. ✓ As linearizability covers sequential consistency, the system is also sequential consistent.

10

TDA 297 -10 5

5 b) He can achieve better availability compared to the three-commit protocol.

(4)

The system remains availble as long as there is one correctly functioning server ✓

The system is sequential consistent and linearizable provided there are no failures ✓

This system with gossip architecture ensures client consistency. The order of executions should respect the real-time ordering of clents requests. Even if the client for example issues a read ~~from~~ to the outdated server. the gossip approach ensures the server will detect the fact that it is outdated and send update request gossip to the other server and completes the update before replying back to the client

How?

11

5

TDA297 -10

c) Constraints:

① Size (W-quorum) ≥ 5 (greater than half of the replicas)

② Size (R-quorum) + Size (W-quorum) ≥ 9 ✓

(The size of the read quorum plus the size of the write quorum must be greater than the total number of replicas)

Availability depends on the size of the quorums. For reading, the smaller the R-quorum is, the higher is the read operation availability. For example, if the read quorum is of size 1, as long as one replica is up and working, the read will succeed. The same goes for the write quorum.

However because of the second constraints, the size of the quorums should be configured based on the system operation frequency (e.g. more read to write and so on).

This configuration will have an impact on the system availability as well.

An example of quorums processing a write, two reads and a write is presented in the next page.

Assume size (write - quorum) = 5
size (read - quorum) = 5

→ Con't

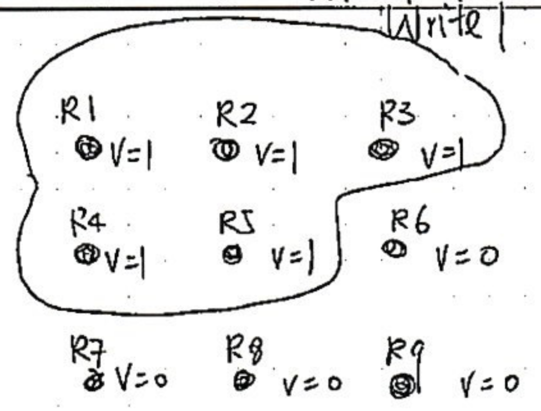
3

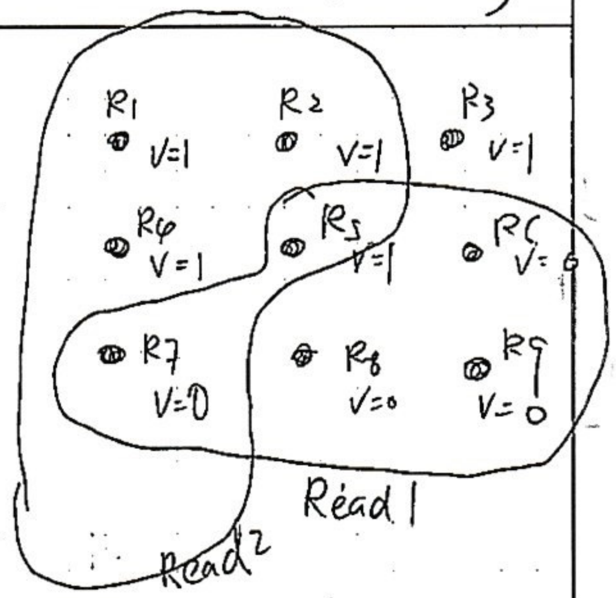
CHALMERS Anonymous code / Anonym kod: TDA 297-10

Points for question (to be filled in by teacher) / Poäng pa uppgiften (fylls i av läraren)

Consecutive page no. / Löpande sid nr: 12

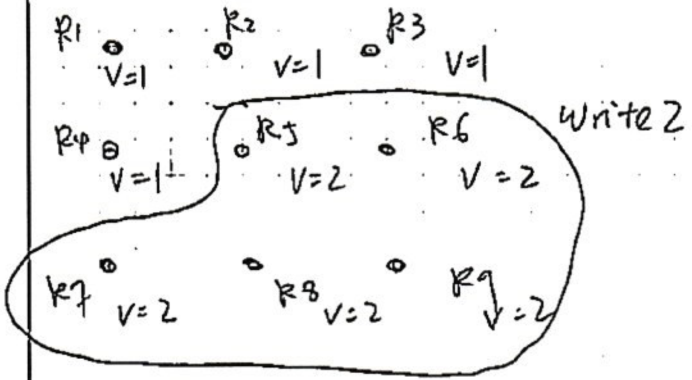
Question no. / Uppgift nr: 5




i. The first write
R1 - R5 are in the write quorum Their verson number becomes 1 while the others are still 0

ii Then two reads are performed as illustrated in the diagram.
Read 1 can read the updated value from R5.
Read 2 can read the updated value from R1 or R2 or R4 or R5




iii The last write is processed. A new write quorum might be constructed but still it ensures more than half of the replicas will have the newest ~~no~~ verson. (R5 - R9).

It is sequential consistent and linearizable
The reason is similar to the prevous systems
As long as there are enough votes for constructing the quorums, the client will always get the most updated read and will manage to write The order of request execations is according to the real-time ordering ~~the~~ of the requests.

3



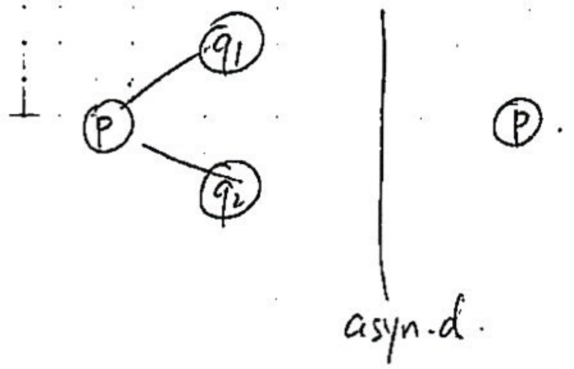
6. The Doorways is a mechanism and separates the processes and execution areas.

Asynchronous doorway:

Entry · $(\forall q \in N(p))$ wait ~~for~~ until $l_{pq} \neq m_1$
Broadcast ("$m_1$: I'm in");

Exit p Broadcast ("$m_2$: "I'm out!");

If it is an asynchronous doorway, the process p that tries to pass will check the neighbours only before p tries to enter.




so as long as p's neighbours have entered and existed, p can enter. p will not be blocked by its neighbours who try to enter again

Synchronous doorway.

Entry wait for $(\forall q \in N(p))$ $l_{pq} \neq m_1$
Broadcast ($m_1$: "I'm in);

Exist p Broadcast ($m_2$: "I'm out);

In the synchronous doorway, p that tries to enter must check the status of all its neighours continously before he actually enters the doorway. It gurantees that if process $P_i$ enters at time t, All the processes that are $\Delta$ away from $P_i$ must enter at $t' < t + \Delta$ →(con 4)

3

| CHALMERS | Anonymous code / Anonym kod | Points for question / Poäng pa uppgiften | Consecutive page no. / Löpande sid nr 14 |
|---|---|---|---|
| | TDA297-10 | | Question no. / Uppgift nr 6 |

Only Asychchronous Doorway + colouring will give us a solution for resource allocation.

It gurantees starvation

Proof sketch:

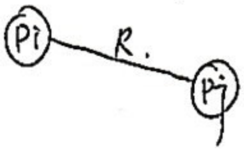
No starvation is guranteed by the ASynchronous doorway As process only checks its neighbours once, the process will not be blocked out of the doorway forever. Eventually all p's neighbours will finish eating and exit, let P enter and not compete with P again.

As long as P enters the doorway, P will get a colour and eventually P will eat.

Mutual Exclusion

proof sketch:

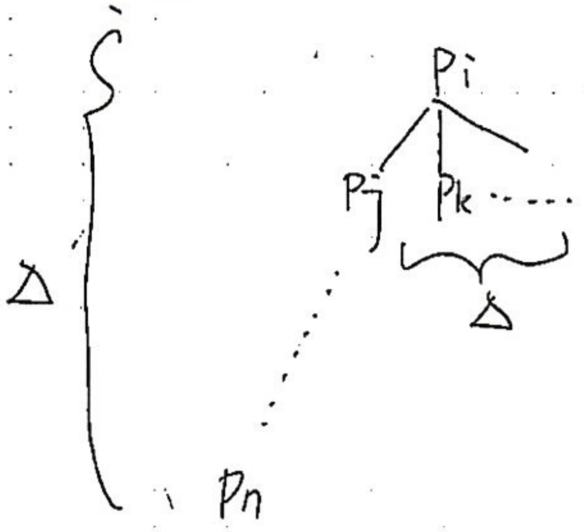
Mutual exclusion is guaranteed by colouring. All adjacent nodes that compete for resources will have different colours and it implies different priority level. The process with lower priority must wait for its neighour who has higher priority to acquire the resource first. Thus, two adjacent nodes will not attempt to assess the resource simultaneously




→ Con't

3

| CHALMERS | Anonymous code / Anonym kod: TDA297-10 | Points for question / Poäng pa uppgiften | Consecutive page no. / Löpande sid nr: 15; Question no. / Uppgift nr: 6 |
| --- | --- | --- | --- |




The time complexity is $O(\Delta^{\Delta})$.

Because the doorway is asynchronized, the process is allowed to become hungry and enters the door one after another. So accessing the resource will behave like a leaf search manner in a tree. As illustrated in the diagram above, Pi might have to wait for $\Delta^{\Delta}$ time before it can acquire the resources

Thus time complexity $O(\Delta^{\Delta})$.