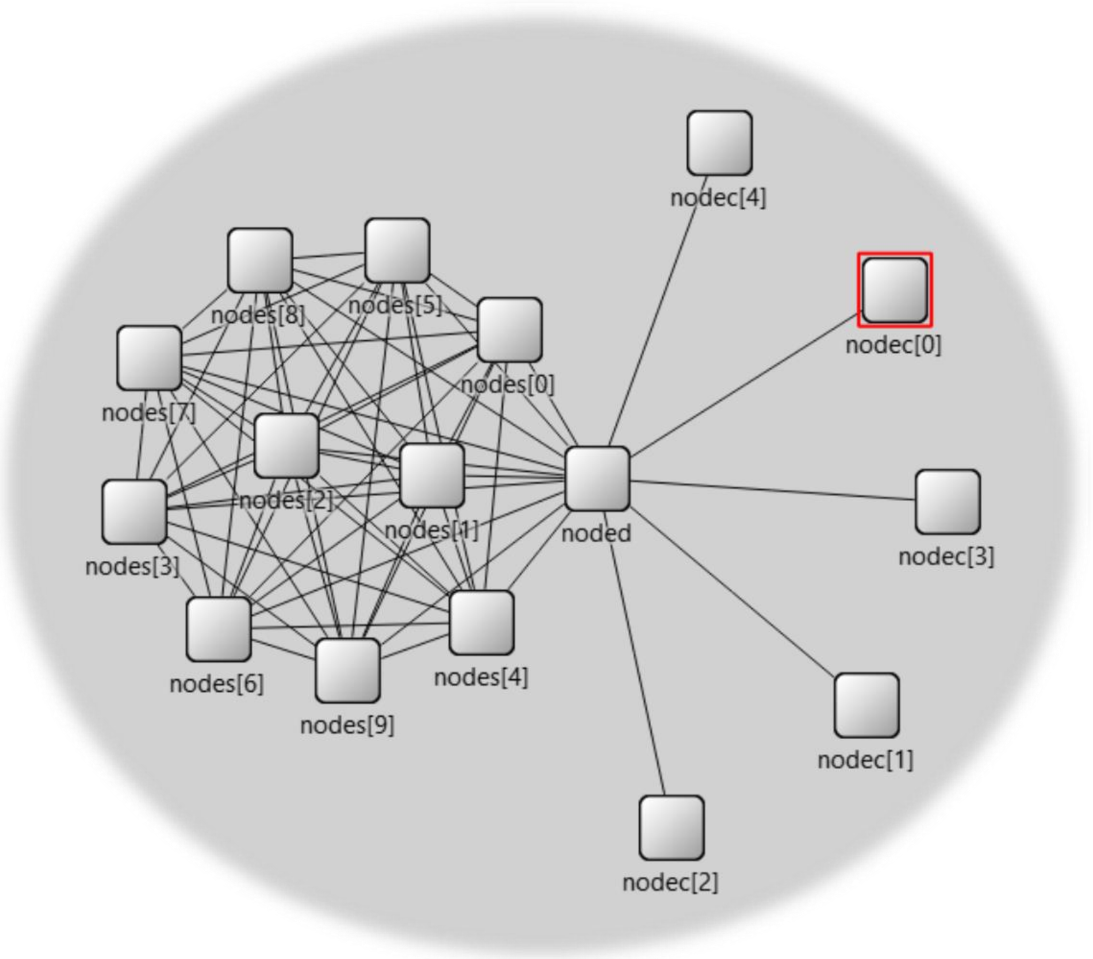
Distributed Systems Project

Transactional key-value store

Fumagalli Stefano

Forbicini Francesca



- System's elements:
  - m server nodes
  - n clients
  - k keys
  - r replicas for each key
  - t operations in every transaction

- And many messages!

ASSUMPTIONS

- Each key and value is a positive integer number
- Each value is initialized with value 0
- Each server knows who is the leader for a key and which servers store any key
- Clients can contact one or more nodes and submit their transactions, one by one
- Processes and links are reliable

How the system works?

- **Client**
  - Sends the transactions to a dispatcher
  - Receives an ACK or Nack after sending the transaction

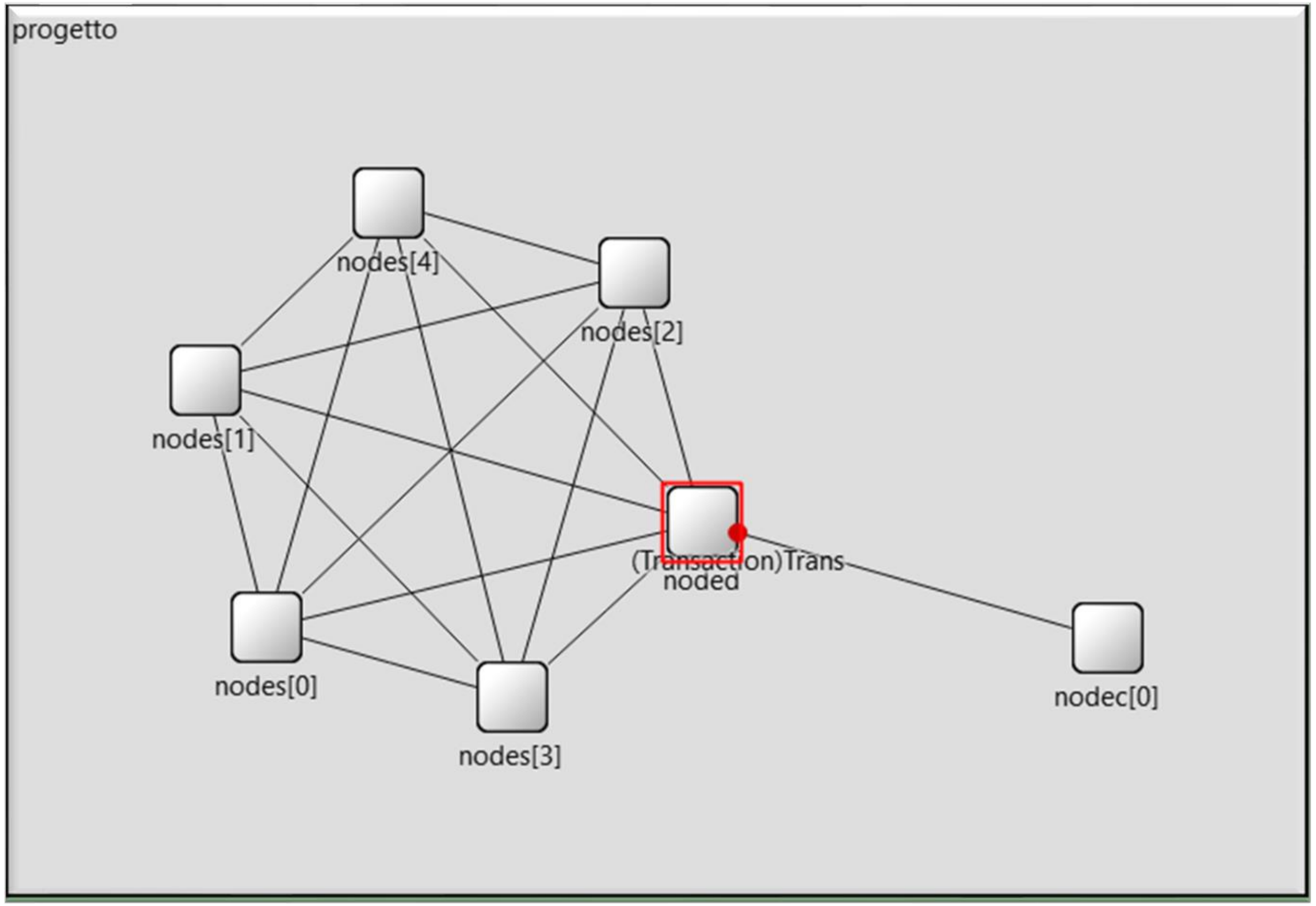
- **Dispatcher**
  - It sends the transactions to a specific server
  - It sends an ACK,NACK or the value read to the client

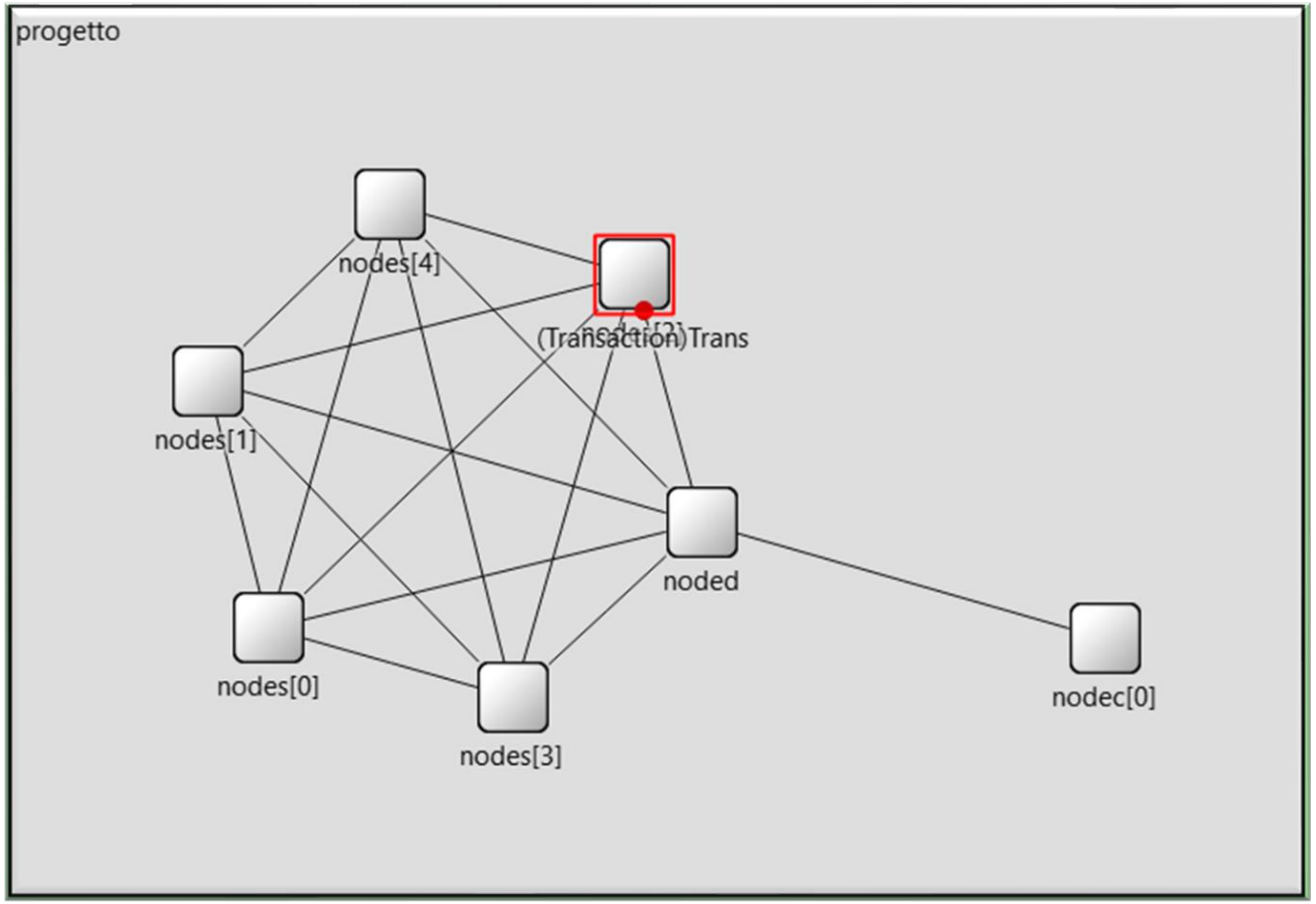
How the system works?

- **ServerNode**
  - Each replicas of the key is contained from server k mod n to (server k mod n) + r
  - It is contacted by the dispatcher
  - If it’s not the leader of the key, it sends the operations to server key mod n
  - It waits for an ACK or NACK to commit or abort the transaction

- **Leader**
  - It’s a server node
  - It receives transactions from other servers
  - It checks if a write or a read can be performed and sends an ACK or NACK

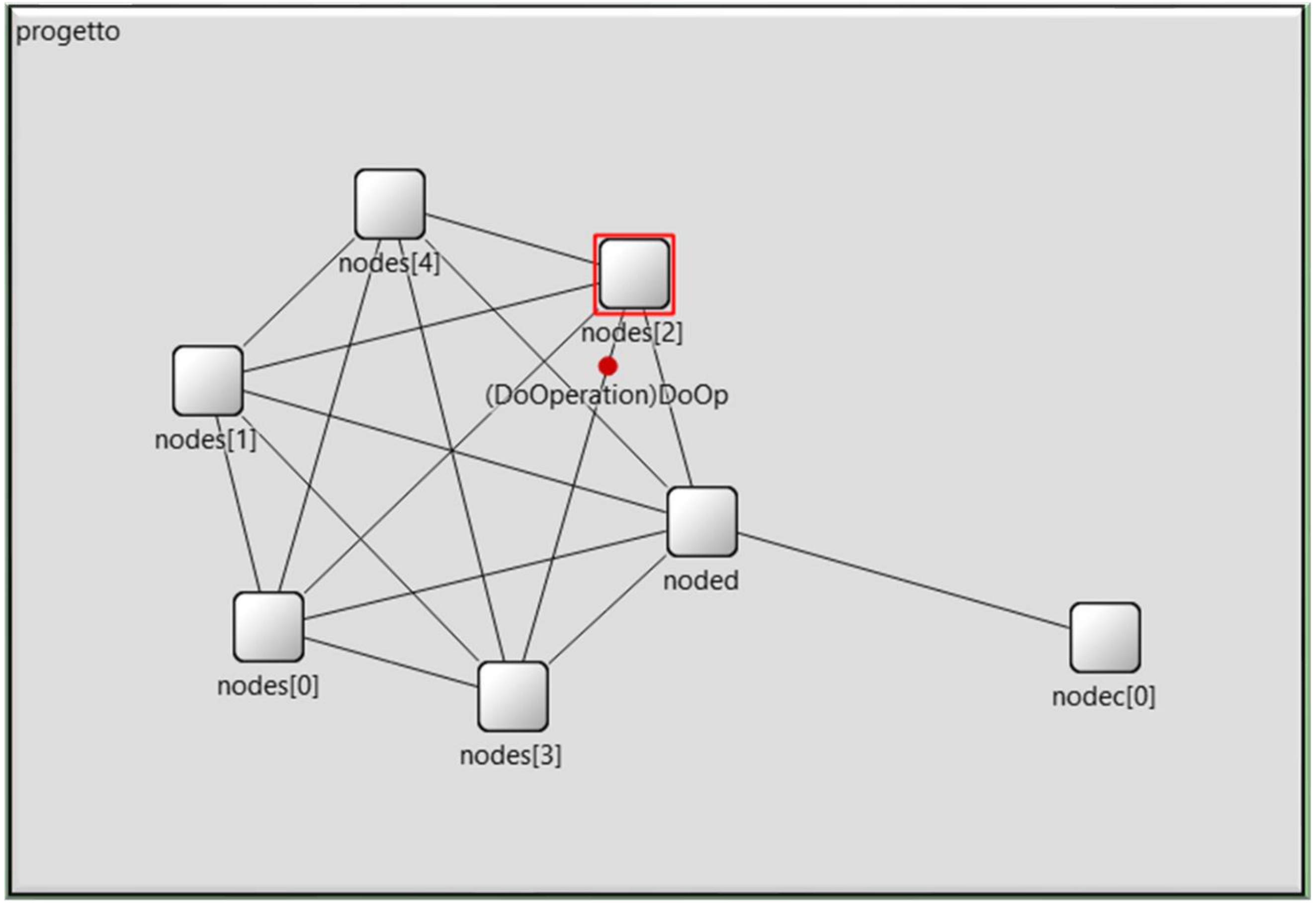
- The client can start the transmission

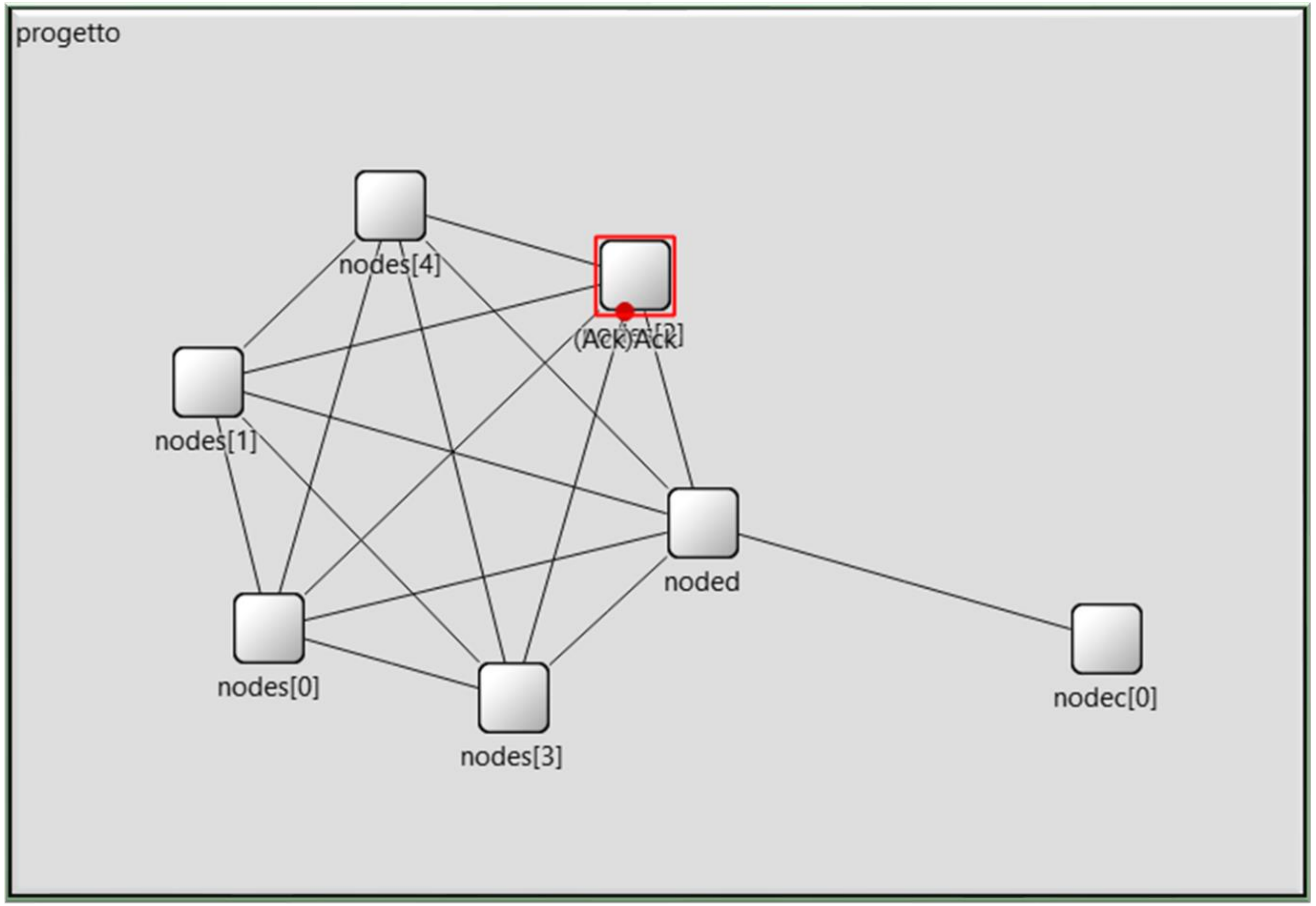




- The dispatcher forwards the transaction to the right server node

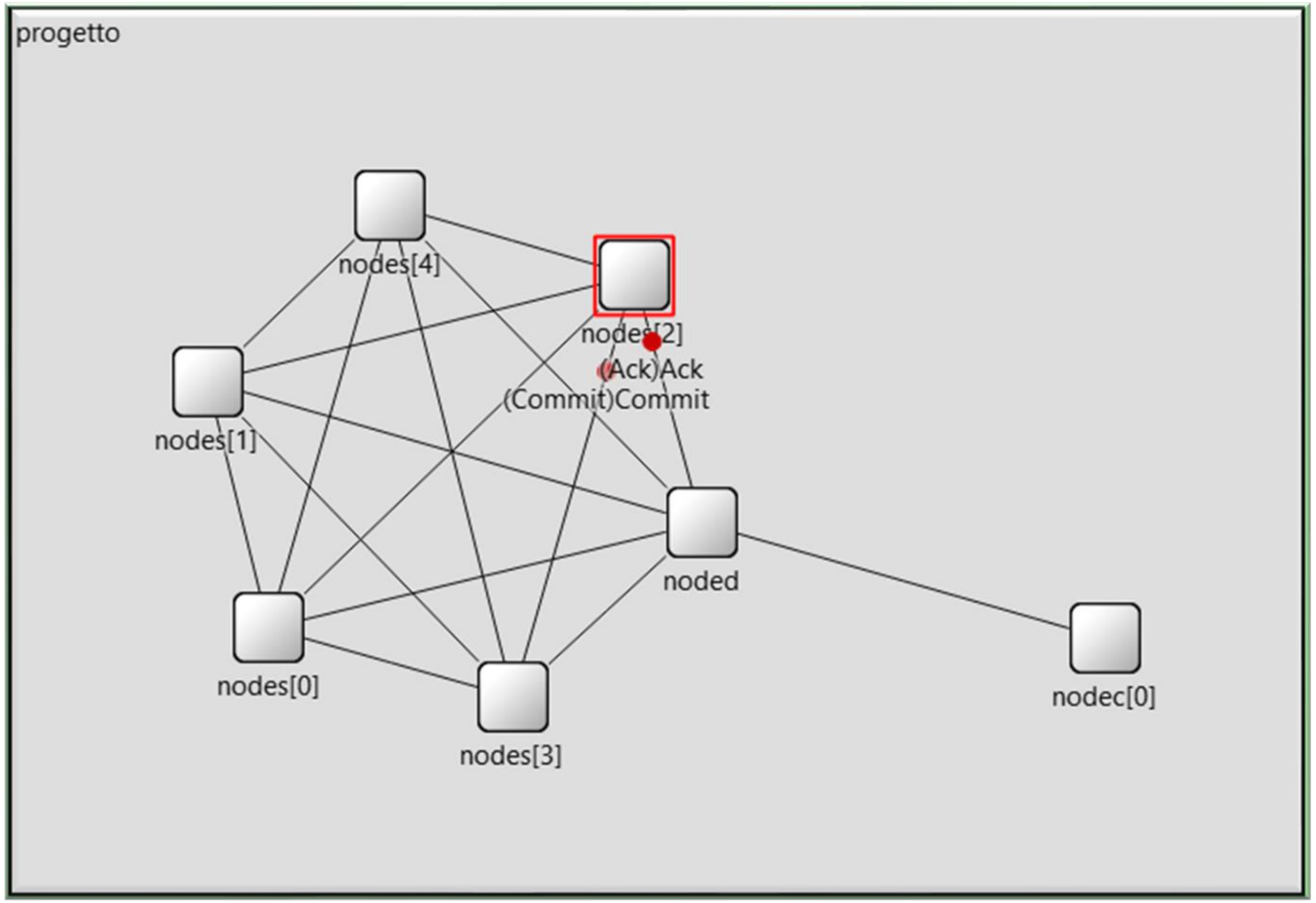
- The server node send a message to the key leader for every operation in the transaction
  - In this case one key is stored by nodes[3] and one by nodes[2]

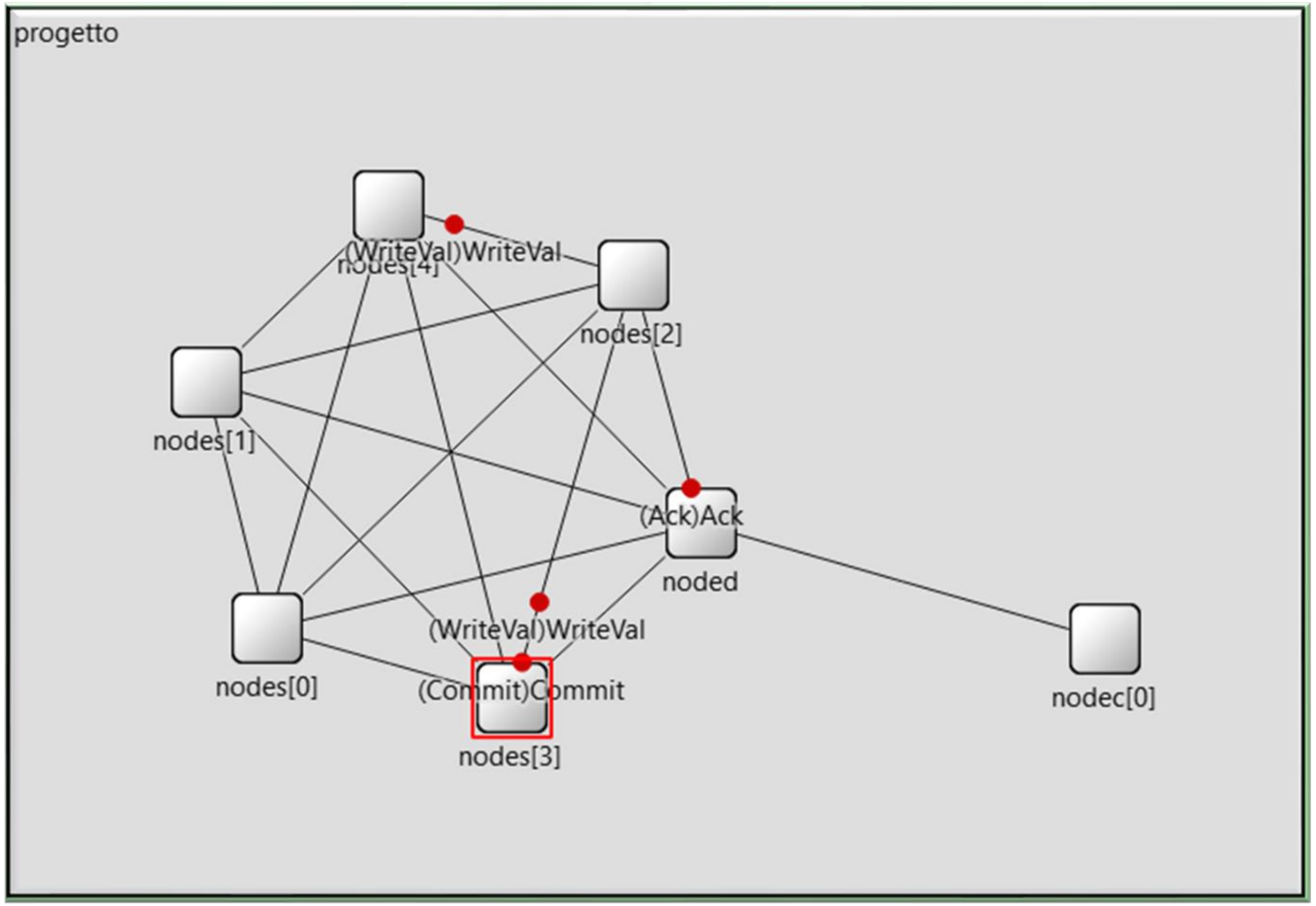




- The nodes[3] sends the ack for the operation

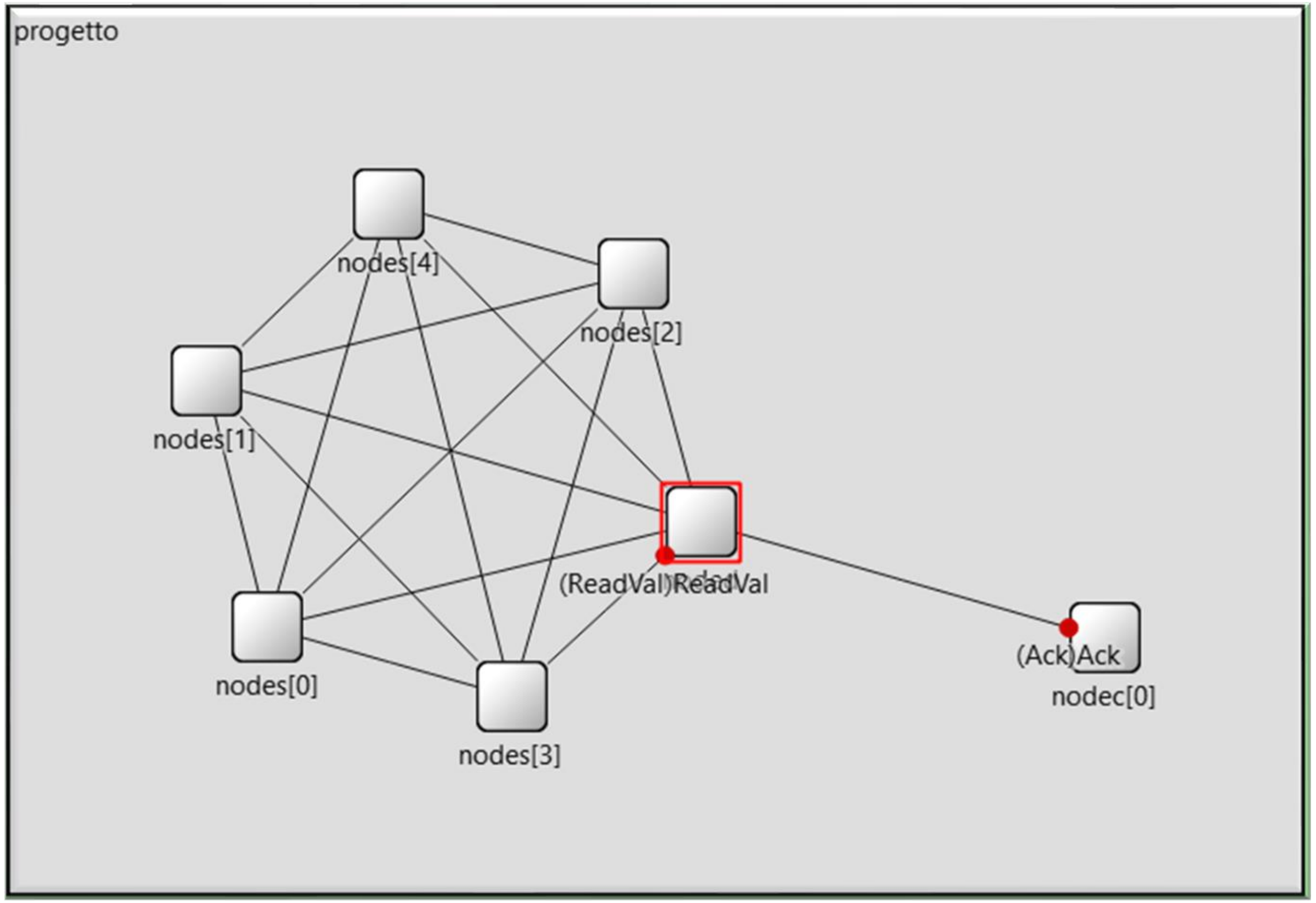
- Since every operation is acknoledged, nodes[2] send:
  - a commit to every node that did an operation
  - An ack to the client, to notify the success of the transaction

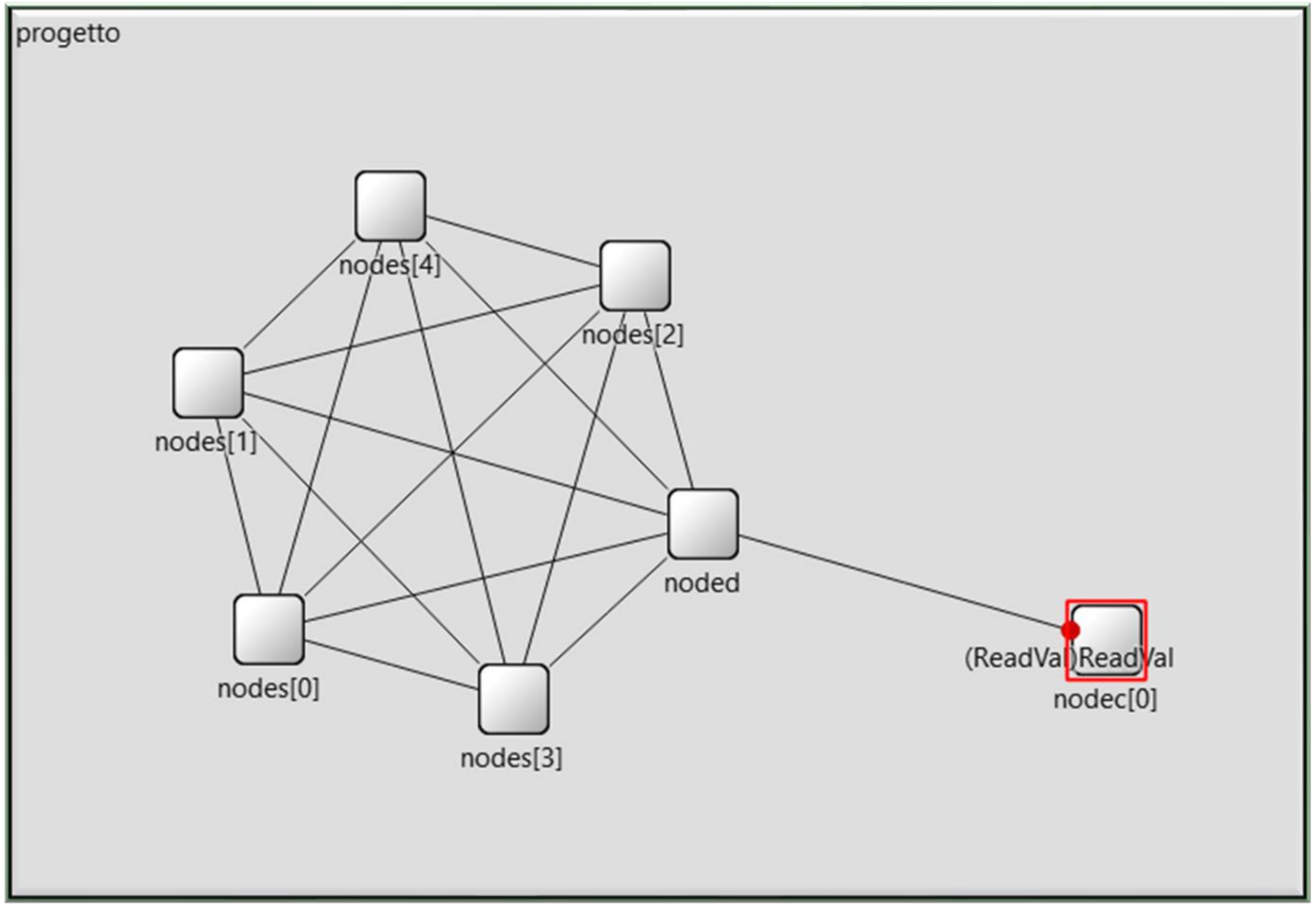




- The operation completed by nodes[2] was a write, it sends a writeVal message to every replicas that store the key to update the value

- The operation completed by nodes[3] was a read, after it receives the commit it sends to the client the readVal message, containing the value read





- The client finally receive the ack and the value read, the transaction is finished

How to handle multiple transaction?

Timestamp ordering

Scheduler receives *read(T,x)* at *time=ts*

- If $ts > ts_{wr}(x)$
  - Let x_{sel} be the latest version of x with the write timestamp lower than ts
  - If x_{sel} is committed perform read on x_{sel} and set $ts_{rd}(x) = max(ts, ts_{rd}(x))$
  - else wait until the transaction that wrote version x_{sel} commits or abort then reapply the rule
- else abort T since the read request arrived too late

When receives *write(T,x)* at *time=ts*

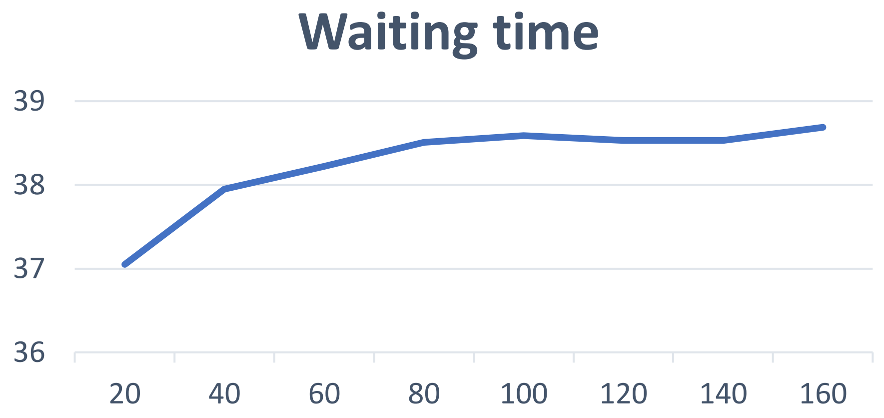
- If $ts > ts_{rd}(x)$ *and* $ts > ts_{wr}(x)$ perform tentative write x_i with timestamp $ts_{wr}(x_i)$
- else abort T since the write request arrived too late

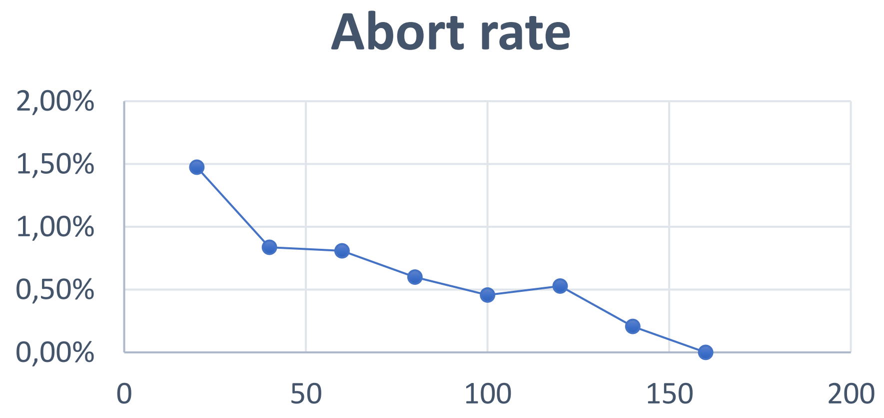
Performance evaluation

- How does the system perform if the number of clients increase?
- What if the transactions contains many operations?
- And if the number of key change?

Increase the number of keys

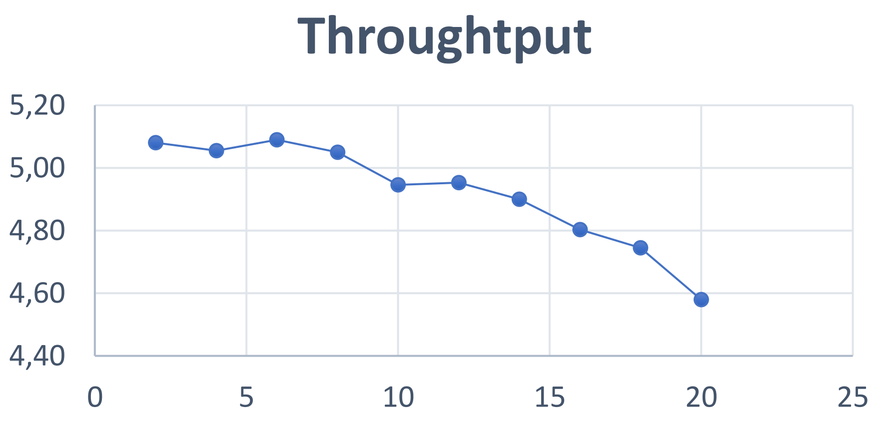
# Key	waitingTime	ackCount	nackCount	Throughput	nack perc
20	37,05	108127	1619	5,01	1,48%
40	37,95	109922	929	5,09	0,84%
60	38,22	109184	889	5,05	0,81%
80	38,51	109454	659	5,07	0,60%
100	38,59	109242	502	5,06	0,46%
120	38,53	109352	580	5,06	0,53%
140	38,53	109402	227	5,06	0,21%
160	38,69	110126	0	5,10	0,00%

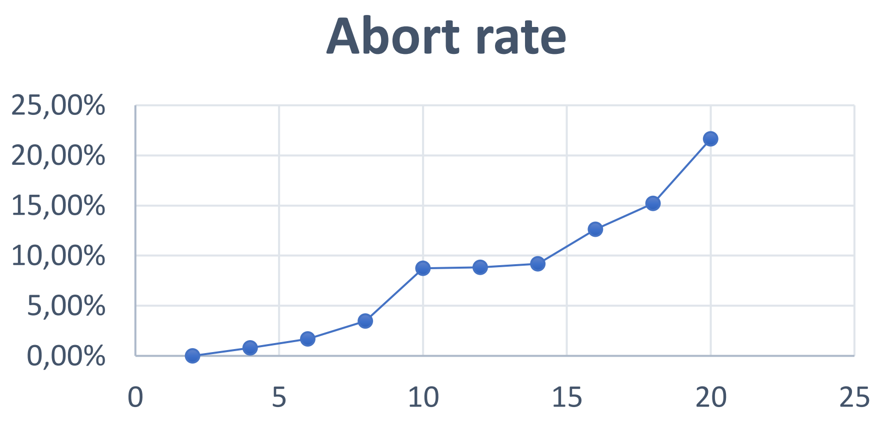




Increase the number of operations in each transaction

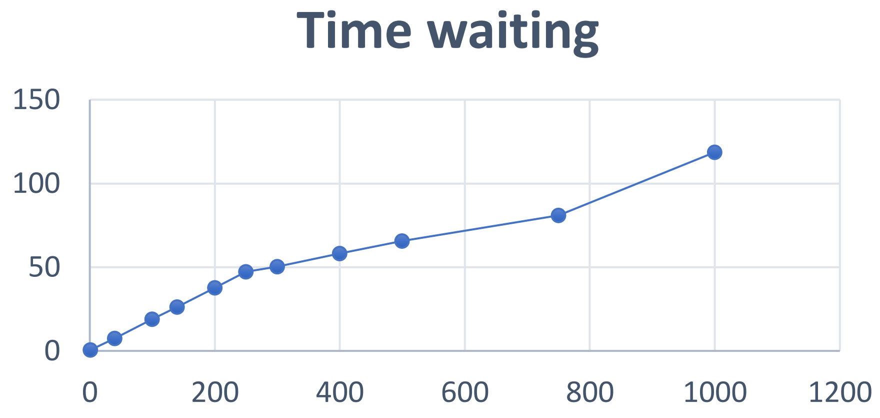
#Op in T	waitingTime	ackCount	nackCount	throughput	nack perc
2	38,83	109740	0	5,08	0,00%
4	38,21	109199	891	5,06	0,81%
6	37,15	109945	1874	5,09	1,68%
8	35,68	109089	3936	5,05	3,48%
10	31,25	106837	10238	4,95	8,74%
12	32,14	106996	10358	4,95	8,83%
14	31,24	105850	10701	4,90	9,18%
16	27,94	103749	14973	4,80	12,61%
18	26,5	102492	18373	4,75	15,20%
20	21,03	98919	27317	4,58	21,64%

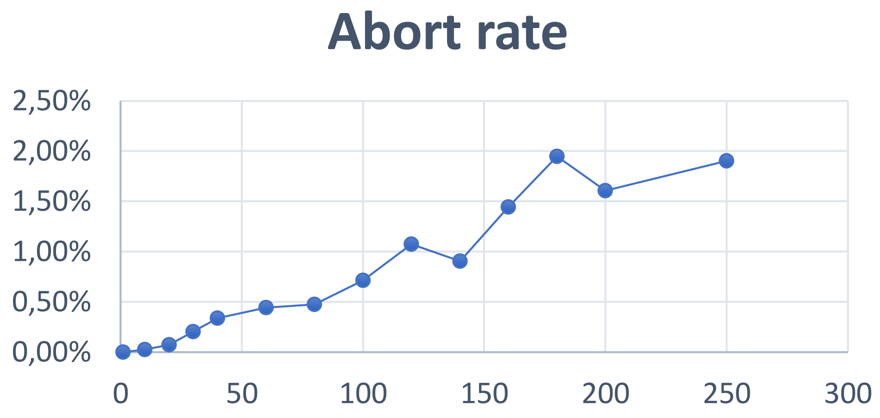




Increase the number of clients

#Clients	waitingTime	ackCount	nackCount	throughput	nack perc
1	0,6	19654	0	0,91	0,00%
10	1,49	107951	26	5,00	0,02%
20	3,45	108013	77	5,00	0,07%
30	5,39	107817	219	4,99	0,20%
40	7,44	108163	367	5,01	0,34%
60	11,22	107941	478	5,00	0,44%
80	15,08	108157	514	5,01	0,47%
100	18,97	108182	778	5,01	0,71%
120	22,59	107997	1172	5,00	1,07%
140	26,27	108224	989	5,01	0,91%
160	30,11	108328	1587	5,02	1,44%
180	33,73	107790	2140	4,99	1,95%
200	37,77	108420	1769	5,02	1,61%
250	47,3	108504	2103	5,02	1,90%
300	50,36	110226	8157	5,10	6,89%
400	58,19	110681	17696	5,12	13,78%
500	65,71	110741	26743	5,13	19,45%
750	81,02	111931	48701	5,18	30,32%
1000	118,63	110781	47199	5,13	29,88%





Change the distribution of the operations

Up to now, all the transactions contain 50% writes and 50% reads

What if we change it?

90% reads

90% writes

50% reads 50 % writes

#Op in T	waitingTime	ackCount	nackCount	Throughput	nack perc
4	38,21	109199	891	5,06	0,81%
8	35,68	109089	3936	5,05	3,48%
12	32,14	106996	10358	4,95	8,83%
16	27,94	103749	14973	4,80	12,61%
20	21,03	98919	27317	4,58	21,64%

90% reads

#Op in T	waitingTime	ackCount	nackCount	Throughput	nack perc
4	38,7	108827	460	5,04	0,42%
8	38,06	109473	1001	5,07	0,91%
12	35,79	109133	3732	5,05	3,31%
16	34,22	107574	6253	4,98	5,49%
20	31,18	106951	10086	4,95	8,62%

90% writes

#Op in T	waitingTime	ackCount	nackCount	thrput ack	nack perc
4	38,62	110322	0	5,11	0,00%
8	36,73	110376	2371	5,11	2,10%
12	34,76	111981	5423	5,18	4,62%
16	29,28	109922	10201	5,09	8,49%
20	25,9	109015	16928	5,05	13,44%