Lab 3

Eventually Consistent Blackboard

Consistency in our Blackboard so far

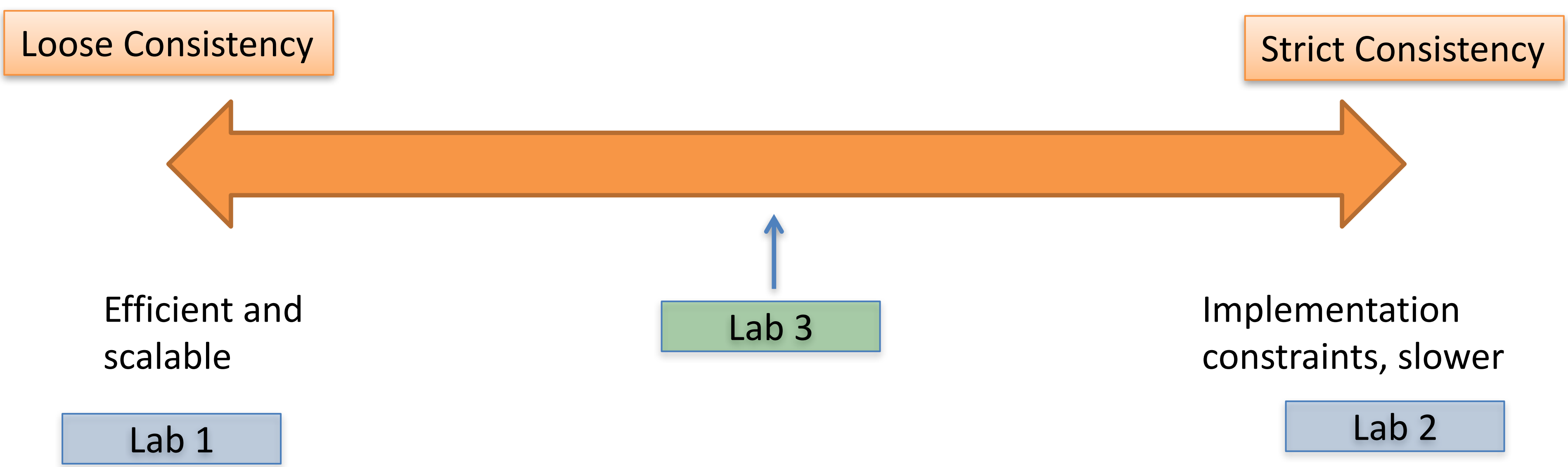
- Simple board (Lab 1):
  - Inconsistent
  - It could happen that different boards show messages in a different order
- With a central leader (Lab 2):
  - Consistent
  - The leader gathered all the messages and decided the order
  - Not scalable
- These are just two extreme cases in the spectrum of consistency trade offs…

Consistency Trade-offs

- Balance between strictness of consistency and efficiency/scalability
  - How "much" consistency we need, depends on the application

Lab 3: Eventual Consistency

- Lab 3 will be somewhere in the middle.
  - Messages can appear in different order **temporarily.**
  - Eventually, they will converge to the same order.



Eventual Consistency

- All replicas *eventually converge* to the same value
- A protocol for eventual consistency:
  - *Writes* are eventually applied in total order
    - same order on all replicas
    - lead to the same value
    - eventual consistency
  - *Reads* might not see most recent writes in total order
- Used in many applications like Google File System (GFS) and Facebook's Cassandra .
- More information on upcoming lectures.

Assumptions/Requirements

- Boards are distributed:
  - No centralized leader, no ring topology
  - You can base on Lab 1’s code (if you want)
- Each post is updated to the local board, then propagated to other boards
- All boards are eventually consistent.
- Delete and modify should also be supported

Sending messages: How-to hints!

- Use logical clocks
  - Each post has a sequence number:
    - Sequence number of a new post:
      the last sequence number received + 1
  - On a node:
    - Posts are ordered by sequence numbers
    - If two posts have the same sequence number, break ties with some rule (e.g. prioritize highest IP address)

Issues with Delete/Modify

- Consider these cases:
  - Two hosts try to delete the same message.
  - A hosts receives an update to <u>delete/modify</u> a message from the blackboard, but that message <u>has not even arrived yet</u> on that host .
    - It can happen in eventual consistency!
- Your algorithm needs to take care of those cases.
  - keep a history of some old updates and act accordingly when new updates arrive.

Task 1: Implement Eventual Consistency

Video/report in which you:

- Demonstrate that your Blackboard is eventually consistent
  - Inconsistent for a while
  - Then becomes consistent
  - Delete and Modify still work
- Briefly discuss pros + cons of this design
- Discuss the cost of your solution, as in Lab 2

Measurements

- You will measure the times it takes for the blackboard to reach consistency.
- Scenario:
  - 4-6-8 number of nodes.
  - 40-50 posts on the blackboard per node (e.g. if N=6 send 40 posts on every node) <u>at the same time</u>
  - Measure the total time to reach consistency state:
    - the longest time among all nodes.
  - We want to see how that time changes as we add more nodes in the system.

Measurements (cont.)

- How to measure?
  - Post messages to nodes at (almost) the same time
    - Write a shell script or a program to automatically post messages
    - Example:

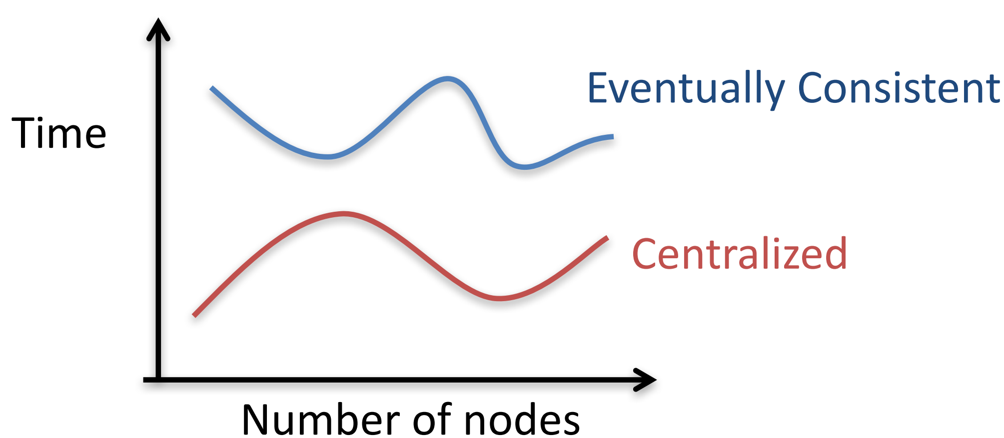
```
for i in `seq 1 250`; do
        curl -d 'entry=t'${i} -X 'POST' 'http://ip:port/entries’
&
done
```

This will post entry=t1, entry=t2, ..., entry=t250 to **one** node

  - Record the time for the Blackboard to reach consistency.
    - Record in a variable the last time you received a request

What to plot

- Do the same measurements for your eventually consistent blackboard and your centralized one from Lab 2.
- Plot the time it takes for all the nodes to get all the messages, as the number of nodes increases:



Task 2: Measurements

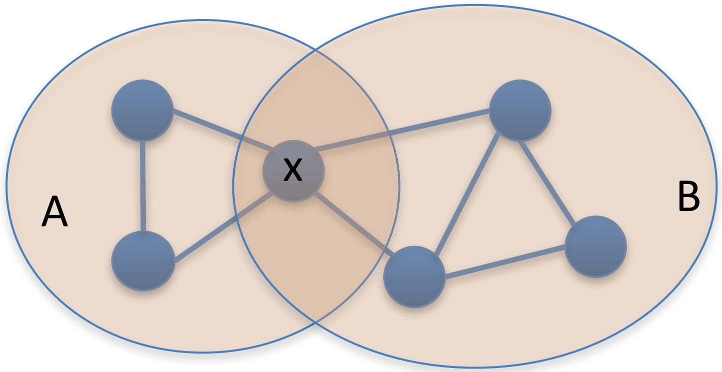
- Run the scenario for Lab2 and Lab3 and take the measurements.
- Use 4, 6 and 8 nodes, to do this modify lab1.py – line 99 nbOfServersPerRegion = 2, then 3, then 4 respectively
- Plot a graph: time to reach consistency as a function of the number of nodes.
- You can use any tool of choice such as openoffice, excel, google docs, matlab, matplotlib …
- Submit the graph, as a pdf file.
- Write a few words explaining the results you see.

Optional: Network Segmentation

- We will try to simulate a scenario where the network is <u>segmented</u> in two parts, then merged again.
- Completely optional, awards up to 5 bonus points.

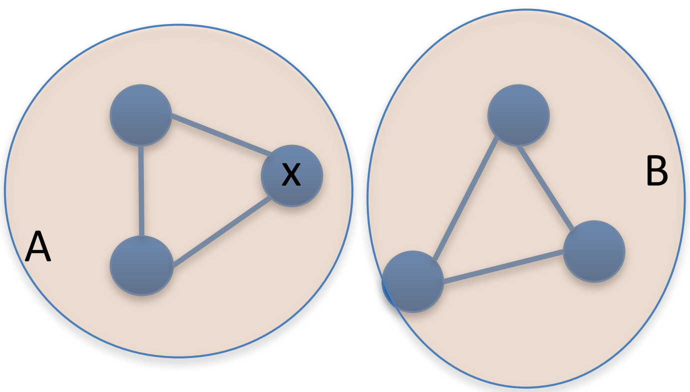
An example

- Phase 1



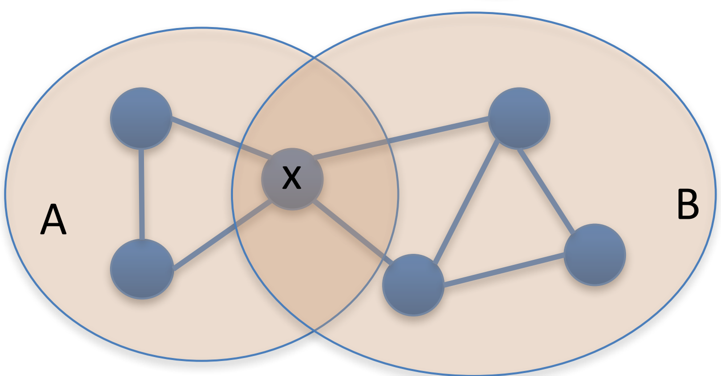
The network is a single segment. Node X communicates with both groups.

- Phase 2



Node X stops communicating with group B. Now there is consistency only within each group.

- Phase 3



Node X starts communicating with group B again. Groups A and B will have inconsistent blackboards unless they exchange their history.

The challenge

- The blackboards must reach consistency when the two disjoined groups are merged together (4 points).
- Note that the topology is no longer “All-to-All”
- You can choose the topology, by having it hardwired in your code, <u>as long as you implement the segmentation</u> in two parts..
- Measure and plot the time it takes to merge, as we add more nodes in the system(1 point).

Optional Task: Network Segmentation

- Video/report: Demonstrate that the network:
  - gets segmented, with every segment having a different view of the blackboard
  - Reaches eventual consistency, when the two segments are merged again.
- Pdf with the graph about the time it takes to merge.
- Briefly discuss how your solution works.

Summary

- Video/Report:
  - The consistent blackboard, show that it works with posts, delete and modify.
  - Optional: Network Segmentation.
- Pdf:
  - The graph comparing consistency time between Lab2 and Lab3.
  - Optional: Graph of the time it takes to merge the segments.