Fault Tolerance

Computer Programs that Can Fix Themselves!

Prof's Paul Sivilotti and Tim Long

Dept. of Computer & Info. Science
The Ohio State University

paolo@cis.ohio-state.edu

Ubiquity of Computers

Distributed Systems

- Computers are increasingly connected
  - Can cooperate to solve a common task
- Example: Voting day
  - Ballots counted in individual polling stations
  - District office sums these counts
  - State office sums the district numbers
- Example: Traffic flow
  - Calculate best route from OSU to downtown Columbus
  - Based on: roads, distances, construction information, traffic jams, *other cars,…*

Distributed Programs: Advantages

- Speed
  - Divide a big problem into smaller parts
  - Solve smaller parts in *parallel*
- Communication
  - Problem is inherently distributed across space
  - Must gather information and coordinate
- Robustness
  - Give the same task to many computers
  - Even if one fails, others are probably ok

So What's the Downside?

- With more computers, there are more things that can go wrong!
  - Despite our best efforts, "faults" occur
- Examples:
  - "blue screen", "segmentation fault", "crash",…
- Possible causes:
  - Environmental conditions
    - Computer unplugged, wireless connection lost
  - Software bug
    - Human error in engineering the software
  - User error
    - Program given bad input, used improperly

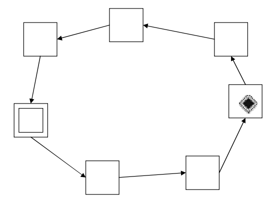
Fault-Tolerant Software

- A program that still does the right thing, *despite faults*
  - Can fix itself, and eventually recover
- For sequential software
  - Restart system
- But for distributed software?
  - Restart isn't practical!
  - System must be "self-stabilizing"

Tokens for Taking Turns

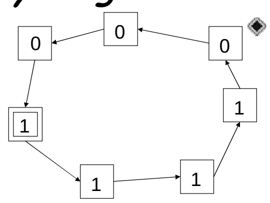
- Consider all the chefs across the city
- Say they need to take turns
  - Only one can be on vacation at a time
- How do they coordinate when to go on vacation?
- Solution: use a “token”
  - Pass token around
  - Rule: *If you have the token, you can go on vacation*

Token Ring



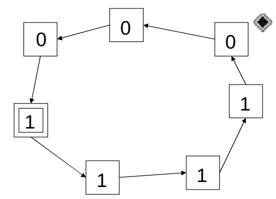
- Problem: What about faults?
  - What happens if token is lost?
  - One fault means disaster!

Prevent Loss of Token: Binary Ring



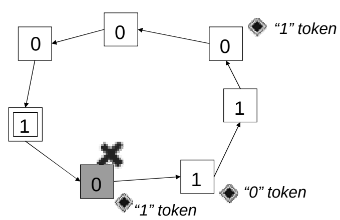
- Rule (for most chefs):
  if left neighbor is different from me, then I have the token
  - Make my number equal to that neighbor’s

Completing the Ring



- One chef is special:
  if left neighbor is same as me, then I have the token
  - Make my number differ from that neighbor’s

Fault: Corruption of Values



- Problem: multiple tokens in ring
  - Tokens chase each other around ring
  - One fault means disaster

k-State Token Ring

- Solution: use more values than chefs!
- Same rule
  - If left neighbor different from me:
    - I have the token! (use it)
    - Change my value to be equal to neighbor
- Again, one chef is special
  - If left neighbor *same* as me
    - I have the token (use it)
    - Change my value to be *one bigger*

Activity: Anthropomorphism

- Form a ring
  - Each person has number cards
  - Each person has a chime
- When you get the token:
  - *Play your chime*
  - <u>Then</u> change your number
- We'll run different versions
  - I'll introduce "faults" and see if you can recover!

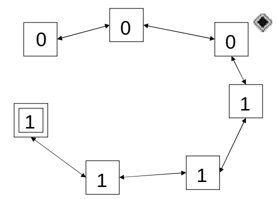
Can We Do Better?

- Problems with this approach?
  1. Consider a *very large* ring
     - Need a *very large* number of states!
  2. Consider a *dynamic* ring
     - When the size changes, everyone needs to be updated!
- A better solution would use a *fixed number* of states
  - Independent of ring size

Constant State Size

- Recall difficulty with binary ring
  - Many tokens in ring, of different "types"
  - Chase each other around the ring, *never colliding*
- Solution A:
  - Force them to "collide" by having more types than chefs
- Solution B?

Don't Use a Ring!



- Solution: chop the ring!
  - Tokens can't circulate
  - They reflect back and forth
  - All chefs responsible for stabilization

4-State Token "Ring"

- Chefs pass tokens right *and left*
  - State = value (0/1) and direction (left/right)
- Left rule (same as before)
  - If left neighbor different from me:
    - I have the token! (use it)
    - Change my value to be equal to neighbor
    - Face right
- Right rule (other direction)
  - If right neighbor same as me and facing me:
    - I have the token! (use it)
    - Face left

Activity 2

- Form a "ring" again
  - Pairs at each station
  - One person for left rule, one for right
- Left rule person:
  - Always "on"
  - Copy value and face right (may already be facing that way)
- Right rule person:
  - Only "on" when you and right neighbor face each other
  - Change direction to face left

Take-Home Messages

- Ubiquity of Distributed Systems
  - Computers are getting smaller, cheaper, faster
  - Increased connectivity
- Faults
  - Some errors are outside of our control
  - Sequential programs can be reset
- Distributed Fault Tolerance
  - Distributed programs that fix themselves
  - Eventually stabilize in spite of faults
- Subtlety of Distributed Algorithms
  - Concurrency and nondeterminism
  - How can we convince ourselves these algorithms are correct?

THE OHIO STATE UNIVERSITY