ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing Group

HS 2009 Prof. Dr. Roger Wattenhofer, Thomas Locher, Remo Meier, Benjamin Sigg

Distributed Systems
Theory exercise 2

Assigned: November 13, 2009
Discussion: November 20, 2009

1 A store

Alex and Bea are working in a huge store. They would like to meet each other, but first they have to agree on a place to meet. Unfortunately, their boss does not like his workers to do anything else but work, so he carefully arranges schedules so that no two workers are ever in the same section.

a) In order to evade their boss's surveillance Alex and Bea came up with an idea. In the fruit corner of the store there is a big basket. They can insert or remove fruit from the basket without their boss noticing. Can they make a decision about a meeting place or not? Why?

b) What if all workers of the store want to meet?

c) The boss has noticed what is going on. Workers are no longer allowed to enter the fruit corner. Unfortunately the boss has no understanding of fruit, so he has to rely on his workers to tell him what to do. Workers are allowed to ask the boss how many fruits of some sort are in the basket, and the boss will replace old fruits with fresh fruits as soon as a worker asks him to do so. Do Alex and Bea have any chance to meet now? Why?

Solution

Is this problem exactly a consensus problem? There are small differences: workers do not have a specific input. Also there is no requirement for the decision to be in the input set, any existing place is a valid result. Could they agree on a specific place before they start working? Yes, but the task would be too easy, so let's assume they did not. Is it important what the initial set of fruit in the basket is? It can influence the decision.

a) They can use the basket like a compare-and-swap register. A protocol could be: Go to the basket. If there are x apples in the basket, then we meet at place x. Otherwise it is undecided and you have to put a valid number of apples into the basket. If the initial set of fruit is not empty, then it might happen that the first person already reads a valid number of apples. In such a case they just meet at a random place, but they do meet.

b) Since CAS has a consensus number of ∞, the exact same protocol works as well.

c) If the basket is initially empty, then basket behaves like an atomic register, thus consensus number is only 1 and they cannot agree on a meeting place.

If the basket is initially not empty, then they might agree on some place. They could use a function on the content of the basket which always returns a valid code for a meeting place (a function like modulo). Notice however that they have no influence on the meeting place.

2 A proof

Why is it not possible for 6 processes to reach consensus if two of them are Byzantine? Sketch a proof.

Solution

We know that consensus for 3 processes, one of them Byzantine, cannot be solved. The proof for this can be found in the lecture notes. We now show that consensus for 6 processes, two of them Byzantine, cannot be solved by means of a contradiction.

Assume that consensus for 6 processes could be solved by an algorithm a. We create a new algorithm b which solves consensus for 3 processes using a.

The algorithm b works as follows: each of the 3 processes (hosts) simulates 2 processes (guests). The input of the guests is equal to the input of their hosts. Correct hosts simulate correct guests, the Byzantine host simulates Byzantine guests. The 6 guests solve consensus using a. The hosts copy the decision of their guests (all guests make the same decision).

This is a contradiction: b cannot work. Since b works if a works, a must be wrong. Hence there exists no algorithm a solving consensus for 6 processes.

3 A graph

In the lecture you learned how to reach consensus in a fully connected network where every process can communicate with every other process. We consider a network that is organized as a 2 dimensional grid such that every process has up to 4 neighbors. The width of the grid is w, the height is h. The grid is big, meaning that $w + h$ is much smaller than $w * h$. While there are faulty and correct processes in the network, we assume that two correct processes are always connected through at least one path of correct processes. In every round processes may send a message to each of its neighbors, the size of the message is not limited.

Hint: unlimited message size allows a process to merge several messages into one big message.

a) Assume there are no faulty processes. Write a protocol to reach consensus. Optimize your protocol for maximum speed.

b) How many rounds does your protocol from a) require?

c) Assume there are $w + h$ faulty processes. The faulty processes may die any time, but may not send wrong messages. In a worst case scenario, how many rounds does the protocol from a) require now?

d) Assume there are $w/2$ Byzantine failures. How could they sabotage your protocol from a)? Only a general idea is required, don not go into details.

A more realistic setting is a network organized as a hypercube. There are $n = 2^m$ processes, each process can communicate with m other processes.

e) Modify the king algorithm so that it works in a hypercube. Optimize the algorithm for maximum resilience. How many failures can the algorithm handle? Assume Byzantine processes can neither forge nor alter source or destination of a message.

f) How many rounds does the algorithm from e) require?

Solution

a) The goal of this protocol is for every process to know the input of all the other processes. Each process internally allocates an array of size $w*h$, each entry of the array represents the input of one process and initially is set to '?'. The process fills the array until no '?' are left, then the process can make its decision.

Whenever a process gets new information, it sends its updated array to all its four neighbors. At the beginning the new information is its own input.

b) The number of required rounds is the length of the longest shortest path between any two processes. In the case of a grid this is $w + h$.

c) The longest shortest path has now a length of $2 * (w + h) + c$ where c is a small constant. So the number of required rounds is $2 * (w + h) + c + 1$. See also figure 1 for an example how the faulty processes can be aligned for producing this long path.

d) Byzantine processes can catch and modify messages. In a setting like in figure 2 only one of four messages reaching t is correct. Process t cannot find out which messages are wrong, thus there is a high probability for t to make false decisions.

e) We only need to alter how broadcast is implemented, all other parts of the king algorithm remain unchanged.

In a hypercube broadcast is not as easy as in a fully connected network. Processes need to forward the messages, and any byzantine process can alter the messages it has to forward.

We can increase the chance for a message to reach its destination unaltered by sending the message over multiple paths. To be more exact: in a hypercube with dimension m we can always find m non-intersecting paths between two processes.

A process may receive the same message up to m times, but the faulty processes could have altered some of them. The process needs to decide what the original message was. Since there is no authentication available, the process can do nothing but assume the message that is most often received is the original message. This leads to $f < m/2$.

f) The king algorithm has $f + 1$ phases, in each phase the algorithm sends three broadcast. So the algorithm requires $3 * (f + 1)$ times the duration of a broadcast.

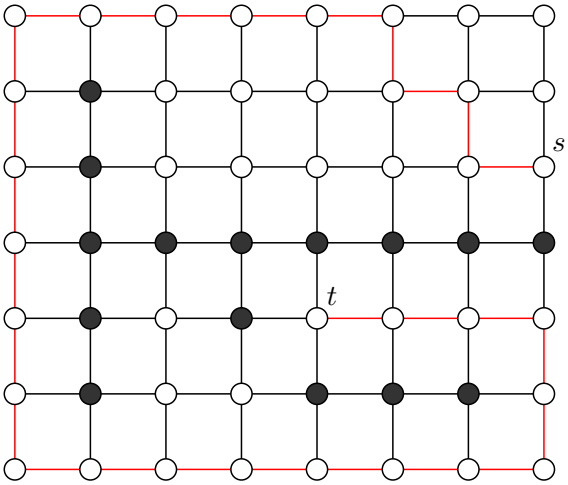

Figure 1: The longest shortest path if $w + h$ processes (dark) are faulty is $2 * (w + h) + c$ where c is a small constant. The path leads from s to t.

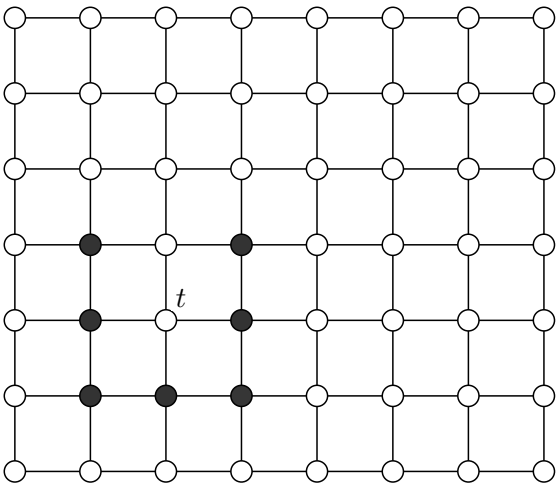

Figure 2: Byzantine processes (dark) can "bombard" process t with wrong information and lead t to a wrong decision

How much time does a broadcast require? Approximately m rounds, because the longest shortest path leading from one corner of the hypercube to the opposite corner has length m.

To be more exact, a broadcast requires $m + 1$ rounds: If process a sends to process b, then b must receive the message up to m times. In a hypercube every process has coordinates which we can express as binary strings of length m. A path between a and b can be seen as sequence of bits that are switched. Example: if $c_a = 000$ and $c_b = 111$ then one path from a to b first visits the processes at 001 (switching bit 3), then the process at 011 (switching bit 2), and finally b (switching bit 1).

The length k of the shortest path between a and b is the number of bits c_a and c_b differ in. There are k non-intersecting paths of length k between a and b. (The i'th path would start with switching the i'th different bit, then the $i + 1$'th different bit, etc.). This leaves $m - k$ paths with a length greater than k.

These $m - k$ paths need first to switch a bit which is the same in c_a and c_b (otherwise the path would intersect with one of the shortest paths). This bit needs to be switched back at the end of a path. Other than that a path needs to switch k bits that differ, leading to $k + 2$ operations all together.

We set k as big as possible, but ensure $k < m$. This means we set $k = m - 1$ and find the longest path a message must ever travel has length $m + 1$.

4 A riddle

The hangman summons his 100 prisoners, announcing that they may meet to plan a strategy, but will then be put in isolated cells, with no communication. He explains that he has set up a switch room that contains a single switch, which is either on or off. It is not known to the prisoners whether the switch initially is on or off. Also, the switch is not connected to anything, but a prisoner entering the room may see whether the switch is on or off (because the switch is up or down). Every once in a while, the hangman will let one arbitrary prisoner into the switch room. The prisoner may throw the switch (on to off, or vice versa), or leave the switch unchanged. Nobody but the prisoners will ever enter the switch room. The hangman promises to let any prisoner enter the room from time to time, arbitrarily often. That is, eventually, each prisoner has been in the room at least once, twice, a thousand times, any number you want. At any time, any prisoner may declare "We have all visited the switch room at least once". If the claim is correct, all prisoners will be released. If the claim is wrong, the hangman will execute his job (on all the prisoners). What's the strategy?

Hint: First try a simplified version of the game. What if the prisoners know the initial state of the switch? What if there are only three prisoners? What if there is a leader?

Solution

Assume that the switch is initially off. One prisoner is the leader. The leader will turn the switch on whenever possible, that is, whenever the switch is off. Any other prisoner will (if possible) turn the switch off, but only the first time it encounters a switch that is on. In other words, the second time a prisoner finds the switch on, the prisoner will leave it on. The leader will (if possible) throw the switch on. After having done that 100 times, the leader

can declare "We have all...". This is because each of the 99 other prisoners have turned the switch off exactly once.

If the initial position of the switch is unknown, we cannot use our simple protocol since we may miscount by one. However, we can easily fix the protocol, by overcompensating this uncertainty of one: We simply let each prisoner turn the switch off twice. Then the leader can safely declare "We have all..." after throwing the switch on 2x99 = 198 times.