Universal Election Protocols

Sec. 3.8

Risto Hakala

February 28, 2007

Outline

- Universal election protocols in general
- Description and analysis of
  1. Mega-Merger
  2. Yo-Yo
- Complexity of the universal election problem
- Summary

Universal Election Protocols

- So far, the election problem has been studied only in specific topologies.
- Universal election protocols are election protocols that do not require a priori knowledge of the network topology.
- Mega-Merger works by constructing a rooted spanning tree.
  - Efficient, but complex in terms of specifications and analysis.
- Yo-Yo is a minimum-finding protocol.
  - Simple to specify and to prove correct, but its real cost is still unknown.
- The restriction **IR** is assumed.

Mega-Merger

Basic principles

- Constructs a rooted spanning tree of the network, where the root is the elected leader in the final spanning tree.
- Rooted spanning trees are merged together until a tree which covers the whole network has been constructed.
- An analogy of cities being merged together is used in the description of the protocol.

Mega-Merger

Concepts and notions

- A city is a rooted tree; the nodes are called districts, and the root is also known as downtown.
- Each city has a level and a unique name; all districts eventually know the name and the level of their city.
- Edges are roads, each with a distinct distance. The city roads are only those serviced by public transport.
- Initially, each node is a city with just one district, itself, and no roads. All cities are initially at the same level.

Mega-Merger

A city

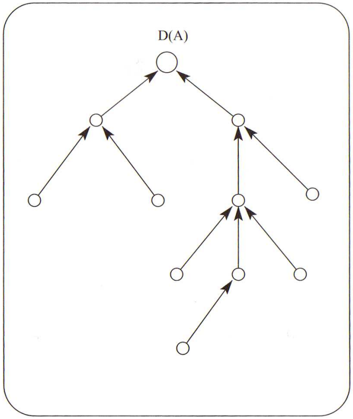


FIGURE 3.49: A city is a tree rooted in its downtown.

Mega-Merger

Merging

- A city A must merge with its closest neighboring city B. To request merging a *Let-us-merge* message is sent on the shortest road $e(A)$ connecting it to B.
- The decision to request for a merger must originate from the downtown and until the request is resolved, no other request can be issued from that city.
- When a merger occurs, the roads of the new city serviced by public transports will be the roads of the two cities already serviced plus the shortest road connecting them.
- The level and name of the new city is adjusted depending on how the merging has been done.

Mega-Merger

Merging (absorption, friendly merger, suspension)

- If $\mathrm{level}(A) = \mathrm{level}(B)$ and $e(A) = e(B)$, then A and B perform a friendly merger.
- If $\mathrm{level}(A) < \mathrm{level}(B)$, A is absorbed in B.
- If $\mathrm{level}(A) = \mathrm{level}(B)$, but $e(A) \neq e(B)$, then the merge process is suspended until $\mathrm{level}(A) < \mathrm{level}(B)$.
- If $\mathrm{level}(A) > \mathrm{level}(B)$, the merge process is suspended: x will locally enqueue the message until $\mathrm{level}(A) \leq \mathrm{level}(B)$.

Mega-Merger

Absorption

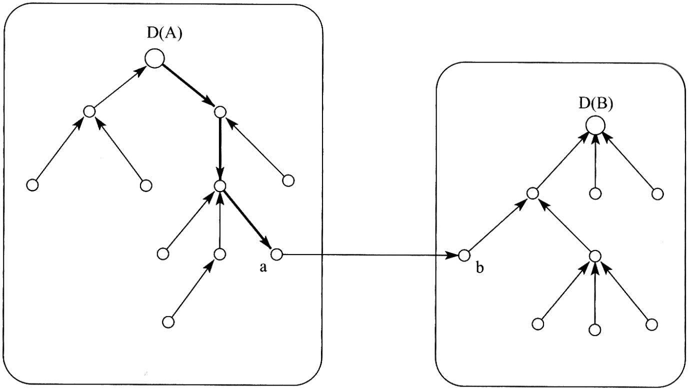
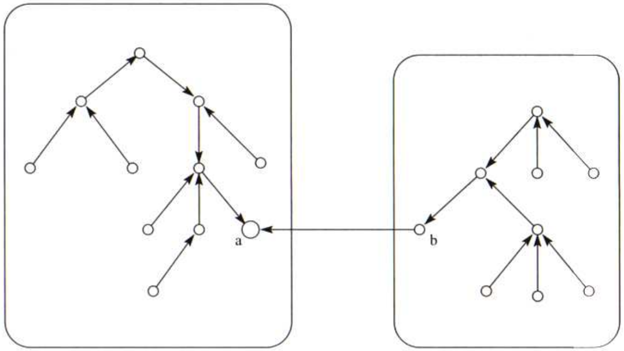


FIGURE 3.50: Absorption. To make the districts of A be rooted in $D(B)$, the logical direction of the links (in bold) from the downtown to the exit point of A has been "flipped."

Mega-Merger

Friendly merger



(b)

FIGURE 3.51: Friendly merger. (a) The two cities have the same level and choose the same merge link. (b) The new downtown is the exit node (*a* or *b*) with smallest id.

Mega-Merger

Choosing the merging edge $e(A)$

1. Each district a_i of A determines the length d_i of the shortest road to another city C
   - For this purpose, an *Outside?* message is sent.
2. The downtown $D(A)$ computes the smallest of all the d_i
   - Same as finding the minimum in a rooted tree.

When determining the length d_i, the district a_i needs not to consider internal roads. However, there may be internal unused roads, which need to be considered, since a_i does not know whether such roads are internal or not.

Mega-Merger

Responding to the *Outside?* message

- If $\mathrm{name}(A) = \mathrm{name}(C)$, c will reply with *Internal* to a_i and the road will be marked as internal. The process is restarted.
- If $\mathrm{name}(A) \neq \mathrm{name}(C)$ and $\mathrm{level}(C) \geq \mathrm{level}(A)$, c will reply with *External* to a_i and d_i will be set.
- If $\mathrm{name}(A) \neq \mathrm{name}(C)$ and $\mathrm{level}(C) < \mathrm{level}(A)$, c will postpone the reply until $\mathrm{level}(C) \geq \mathrm{level}(A)$.

This completes the specification of Mega-Merger.

Analysis of Mega-Merger

Progress and deadlock

- Complex scenarios can occure due to concurrency, postponements, and communication delays.
- Because of this complexity, there is no satisfactory complete proof of the correctness of the Mega-Merger protocol.
- It can be proven, however, that Mega-Merger
  1. is deadlock free and ensures progress; and
  2. correctly elects a leader.

Analysis of Mega-Merger

Cost

- The total cost of Mega-Merger can be determined considering the number of levels, the number of messages per level, and the number of useless messages.
- The total number of messages is

  $$\mathbf{M}[\mathit{Mega\text{-}Merger}] \leq 2m + 5n\log n + n + 1,$$

  where m is the number of links and n is the number of nodes.
- In fact, Mega-Merger is worst case optimal, i.e., $O(m + n\log n)$.

Analysis of Mega-Merger

Minimum-cost spanning trees

- Minimum-cost spanning trees are important for determining, e.g., the spanning tree where broadcasting is the cheapest.
- Mega-Merger creates minimum-cost spanning trees even when the links have no values asigned to them.
- The design complexity of Mega-Merger is its main drawback, since it makes any actual implementation difficult to verify.

Yo-Yo

Main principles

- Yo-Yo is a minimum-finding algorithm.
- Unlike the previous protocol, Yo-Yo has simple specifications, and its correctness is simple to establish.
- The Yo-Yo algorithm consists of two parts: a preprocessing phase and a sequence of iterations.
- Each iteration is composed of phases YO- and -YO.

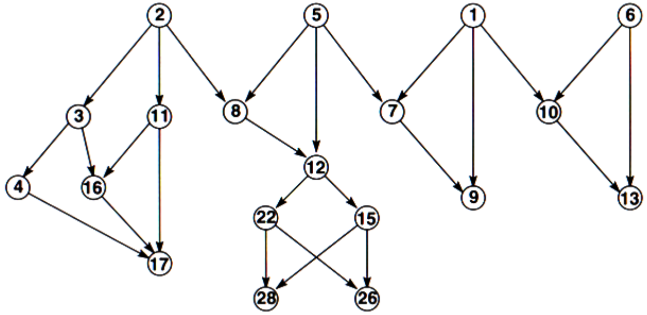
Yo-Yo

The setup phase

- Every entity x exchanges its id with its neighbours.
- Then, x will logically orient each incident link (x, y) in the direction of the entity with the largest id.
- This results in a directed acyclic graph (DAG), where there are three types of nodes:
  - source is a node where all the links are out-edges.
  - sink is a node where all the links are in-edges.
  - internal node is a node, which is neither a source nor a sink.
- The source nodes are the candidates for the minimum in the iteration phase, where iterations remove candidates from consideration.

Yo-Yo

A directed acyclic graph



(b)

FIGURE 3.53: In the Setup phase, (a) the entities know their neighbors' ids and (b) orient each incident link toward the smaller id, creating a DAG.

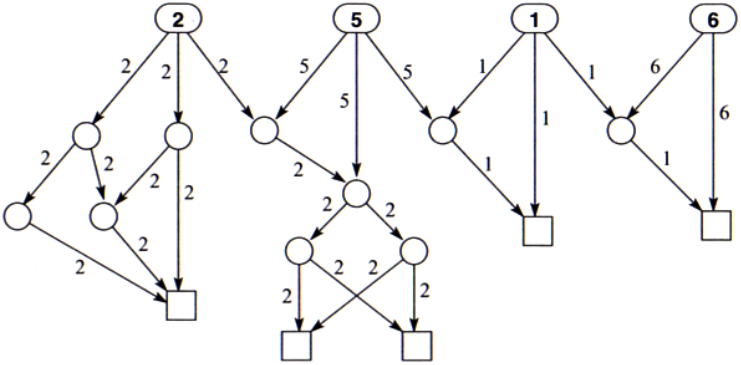
Yo-Yo

Iteration (YO- phase)

- This phase is started by the sources and its purpose is to propagate to each sink the smallest among the values of the sources connected to that sink.
  1. A source sends its value down to all its out-neighbours.
  2. An internal node waits until it receives a value from all its in-neighbours. It then computes the minimum of all received values and sends it down to its out-neighbours.
  3. A sink waits until it receives a value from all its in-neighbours. It then computes the minimum of all received values and starts the second part of the iteration.

Yo-Yo

Iteration (YO- phase)



(a)

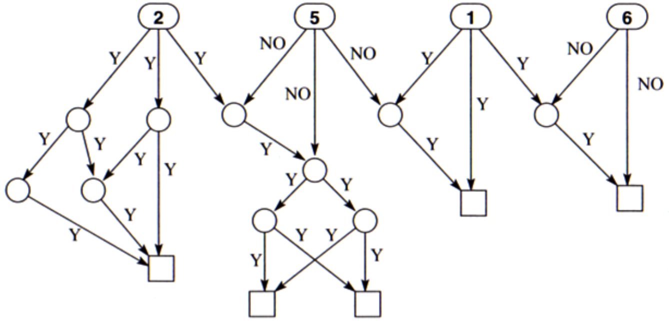
Yo-Yo

Iteration (-YO phase)

- This phase is started by the sinks and its purpose is to eliminate some candidates by transforming some sources into sinks or internal nodes.
  4. A sink sends YES to all in-neighbours from which the smallest value has been received. It sends NO to all others.
  5. An internal node waits until it receives a vote from all its out-neighbours. If all votes are YES, it sends YES to all in-neighbours from which the smallest has been received and NO to all the others; otherwise, it sends NO to all its in-neighbours.
  6. A source waits until it receives a vote from all its out-neighbours. If all votes are YES, it survives; otherwise, it is no longer a candidate.

Yo-Yo

Iteration (-YO phase)



(b)

FIGURE 3.54: In the Iteration stage, only the *candidates* are sources. (a) In the YO- phase, the ids are filtered down to the sinks. (b) In the -YO phase, the votes percolate up to the sources.

Yo-Yo

Iteration (-YO phase)

- Directions of the links must be modified so that only the sources that are still candidates will still be sources.
  7. When a node sends NO to an in-neighbour, it will reverse the direction of that link.
  8. When a node receives NO from an out-neighbour, it will reverse the direction of that link.

The iteration phase is repeated until there is only one candidate left.

Yo-Yo

Pruning

- Applying an iteration to a DAG with more than one source will result into a DAG with fewer sources. The source with smallest value will still be a source.
- Therefore, the source with the smallest value will be the only one left under consideration.
- Additional mechanisms are needed for the smallest node to distinguish that the process shoud end.
- Pruning is used to ensure the termination of the process.

Yo-Yo

Pruning

- The purpose of pruning is to remove from the computation, nodes and links that are useless.
- Pruning is achieved through the following rules:
  9. If a sink is a leaf, then it is useless and asks its parent to be pruned.
  10. In in the YO- phase, a node receives the same value from more than one in-neighbour, it will ask all of them except one to prune the link connecting them.
- Pruning is performed during the -YO phase by declaring links useless.
- If the DAG has a single source, then the new DAG is composed of only one node, the source, after an iteration.

Analysis of Yo-Yo

Costs

- Without pruning, the total cost of Yo-Yo is

$$\mathbf{M}[\textit{Yo-Yo (without pruning)}] \leq 2m \log n + l.o.t.$$

- In other words, the total cost is $O(m \log n)$ messages.
- The real cost (with pruning) is an open research problem.

Complexity of the Universal Election Problem

- What is the complexity of the election problem in general?
- It can be shown that, under **IR**, constructing a spanning tree **SPT** and universal election **Elect** are computationally equivalent

  $$\mathbf{Elect}(IR) \equiv \mathbf{SPT}(IR)$$

  but also that they have the same complexity:

  $$\mathcal{M}(\mathbf{Elect/IR}) = \mathcal{M}(\mathbf{SPT/IR}).$$

Summary

- Universal election protocols work without knowledge of the network topology.
- Mega-Merger is an efficient protocol that constructs a rooted spanning tree of the network, where the root is the elected leader.
- Yo-Yo is a minimum-finding protocol with a more simple specification than Mega-Merger.
- Under **IR**, constructing a spanning tree and universal election are computationally equivalent.