Distributed Coordination

Jonathan Geisler

May 10, 2006

Taylor University
CSS

Distributed Algorithms

- Information spread between multiple processors

Distributed Algorithms

- Information spread between multiple processors
- Processors may only make decisions based on local information

Distributed Algorithms

- Information spread between multiple processors
- Processors may only make decisions based on local information
- Avoid single points of failure

Distributed Algorithms

- Information spread between multiple processors
- Processors may only make decisions based on local information
- Avoid single points of failure
- **Global time is unavailable!**

Event Ordering

Some tasks have strict ordering requirements (e.g., resource use must follow its allocation). The last property from the previous slide makes this a difficult requirement to fulfill.

Leslie Lamport's logical clock

Leslie took the ideas from Einstein's theory of relativity and applied them to this problem. He started by defining a transitive relationship ($\rightarrow$) for time dependence (i.e., if A happens prior to B, then $A \rightarrow B$). If there is no timing relationship, the two tasks are said to be *concurrent*.

Rules for the logical clock

- Messages imply an ordering between sender and receiver

Rules for the logical clock

- Messages imply an ordering between sender and receiver
- Each process keeps track of its local event ordering

Rules for the logical clock

- Messages imply an ordering between sender and receiver
- Each process keeps track of its local event ordering
- When a process sends a message, it includes its local logical time. The receiver will update its local logical time if it is too early.

Rules for the logical clock

- Messages imply an ordering between sender and receiver
- Each process keeps track of its local event ordering
- When a process sends a message, it includes its local logical time. The receiver will update its local logical time if it is too early.
- Actual time is not as important as the ordering in this system!

Rules for the logical clock

- Messages imply an ordering between sender and receiver
- Each process keeps track of its local event ordering
- When a process sends a message, it includes its local logical time. The receiver will update its local logical time if it is too early.
- Actual time is not as important as the ordering in this system!
- If an event depends on something else, it will either run on the same processor or wait for a message to arrive.

Mutual exclusion

There are three basic options:

1. Lock manager

Mutual exclusion

There are three basic options:

1. Lock manager
2. Token passing

Mutual exclusion

There are three basic options:

1. Lock manager
2. Token passing
3. Fully distributed

Mutual exclusion

There are three basic options:

1. Lock manager
2. Token passing
3. Fully distributed
   - Request lock from all other processes (with timestamp)

Mutual exclusion

There are three basic options:

1. Lock manager
2. Token passing
3. Fully distributed
   - Request lock from all other processes (with timestamp)
   - Respond to all messages you don't care about

Mutual exclusion

There are three basic options:

1. Lock manager
2. Token passing
3. Fully distributed
   - Request lock from all other processes (with timestamp)
   - Respond to all messages you don't care about
   - Resolve conflicts via timestamps

Atomic actions

The key is to use transactions to ensure multiple actions succeed or fail as a unit. The traditional two phase commit (see COS 341 for details) is used for implementation.

Concurrency Control
(i.e., handling shared data at multiple locations)

It is trivial to deal with this situation if the data only exists at one location because that location can monitor its data using techniques we've already looked at. What do we do when data is replicated?

- Have a single site manage all the locks

Concurrency Control
(i.e., handling shared data at multiple locations)

It is trivial to deal with this situation if the data only exists at one location because that location can monitor its data using techniques we've already looked at. What do we do when data is replicated?

- Have a single site manage all the locks
- Require a majority of data holders to agree to a lock

Concurrency Control
(i.e., handling shared data at multiple locations)

It is trivial to deal with this situation if the data only exists at one location because that location can monitor its data using techniques we've already looked at. What do we do when data is replicated?

- Have a single site manage all the locks
- Require a majority of data holders to agree to a lock
- Favor shared locks to exclusive locks

Concurrency Control
(i.e., handling shared data at multiple locations)

It is trivial to deal with this situation if the data only exists at one location because that location can monitor its data using techniques we've already looked at. What do we do when data is replicated?

- Have a single site manage all the locks
- Require a majority of data holders to agree to a lock
- Favor shared locks to exclusive locks
- Give one data holder the exclusive rights to the data

Deadlock avoidance/prevention

Slight modifications to the single processor schemes can be made to allow distributed processing to avoid/prevent deadlock. The three primary methods discussed in your book are:

1. Global resource ordering
2. Wait-die
3. Wound-wait

Deadlock prevention

Again, we can slightly modify the single processor schemes to detect deadlock in a distributed system. We'll still use our resource allocation graph, but its representation is where the change lies. We can either store it on a single computer or distribute it throughout the processors that control the resources

Election algorithms

For some of the methods mentioned, we discussed a single coordinator. We also mentioned the likelihood of failure. How do we handle the coordinator failing? The main options are:

- Bully algorithm
- Ring algorithm

Byzantine generals problem
(i.e., reaching agreement in a distributed world)

It turns out having processes agree with each other what they are both doing is difficult. You book demonstrates that it is **impossible** in the presense of unreliable communication, and gives the non-trivial conditions that must exist in the presense of faulty processes.