Highly available, Fault tolerant Co-scheduling System

With working implementation

Center for Computation & Technology
AT LOUISIANA STATE UNIVERSITY

Bringing it all Back Home

Acknowledgements

- This work isn’t part of an official project, but it isn’t all my own work!

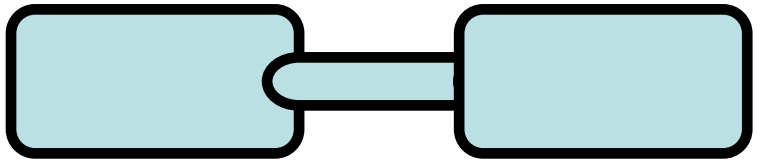
- Much of the design of this work, and in particular the idea to use the Paxos Consensue algorithm in the first place, are entirely due to Mark Mc Keown from Manchester.

Co-scheduling

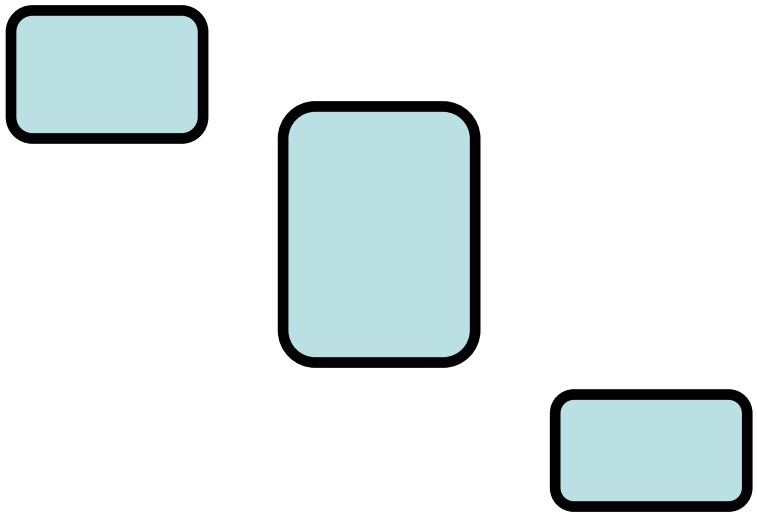
- Most obvious definition is scheduling a number of resources for the same time
- However, I think of it as meaning the scheduling of multiple resources, where the scheduling is done in an atomic fashion - all resources or none are booked
- The resources might be required for different times...

Examples

- Schedule 8 procs on Helix, 32 on Peyote, plus optical network for 12pm to 1pm



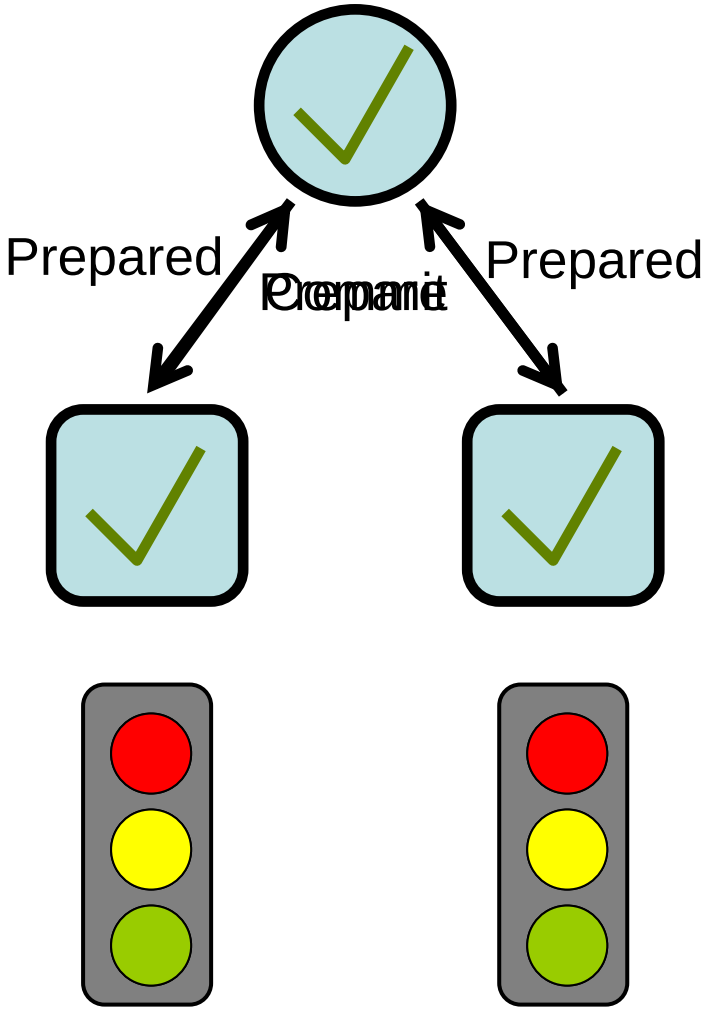
- Schedule workflow tasks onto machines X, Y and Z to start at 1pm (1 hour), 1.30pm (3 hours), 5pm (1 hour)



To Have Phased Commit...

Can have phased commit:

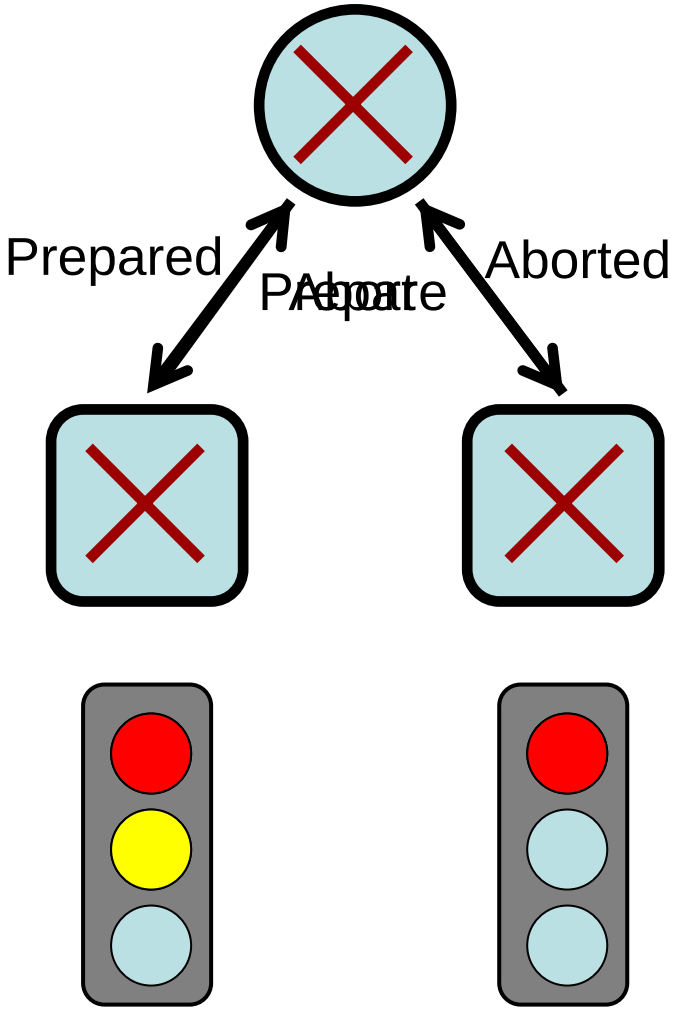
1. resources are asked for a reservation (prepare)
2. resources can say yes/no (prepared/aborted), where yes is a commitment to do the work
3. Then:
   a) If all resources return prepared, then they will be told to finalise the reservations (commit)
   b) Otherwise, all resources are told to forget the reservations (abort)



To Have Phased Commit...

Can have phased commit:

1. resources are asked for a reservation (prepare)
2. resources can say yes/no (prepared/aborted), where yes is a commitment to do the work
3. Then:
   a) If all resources return prepared, then they will be told to finalise the reservations (commit)
   b) Otherwise, all resources are told to forget the reservations (abort)



...Or To Not Have Phased Commit

Can do without this, using WS-Agreement:

1. You make individual reservations with resources
2. If a resource says no, cancel the others

Is Phased Commit Needed?

- Long debates on GRAAP mailing list about this between me & Karl (agreed to differ!)
- Without phased commit, you make a lot of assumptions about the RMs.
  - May end up with some reserved, some not, e.g. if there is some failure in the network
  - You end up making & breaking reservations - the end resources don't know it's coscheduling.
  - You may hit a "reservation quota"
  - You may get charged for making a reservation
  - You may get a decreasing rate of response (unreliable client who keeps canceling jobs!)
  - RMs may even think this is some denial of service attack

Problems with 2-phase Commit

- The transaction manager is a single point of failure
- If it fails/goes away the user/RMs may not be able to discover the outcome
- Not good in a distributed environment, without reliable message delivery

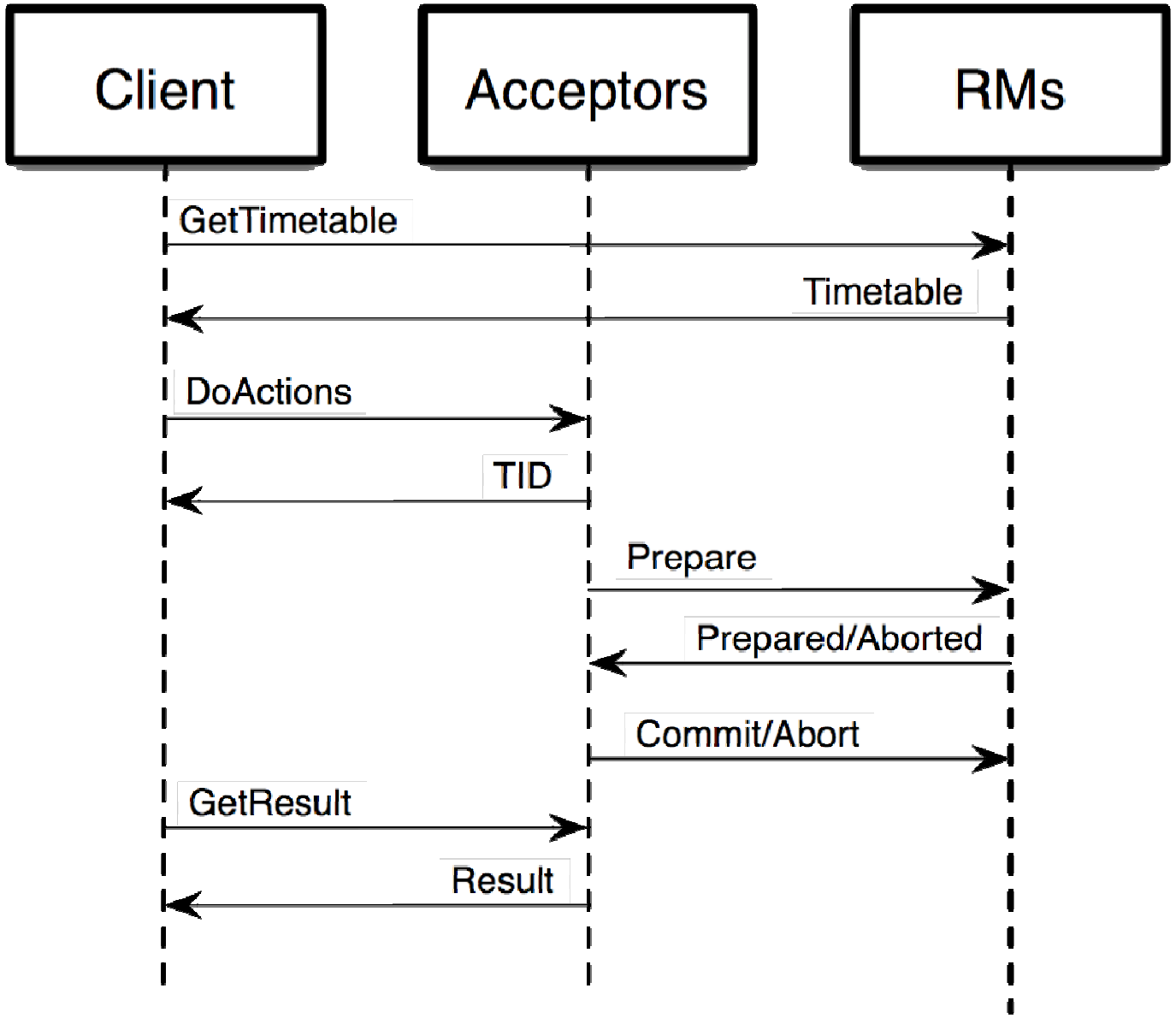
Paxos Consensus

- Leslie Lamport's (in)famous algorithm
- Hard to learn, but ultimately simple
- Maybe even obvious, inevitable
- Best formulation: "Paxos made Simple"

- Was applied to Transaction Commit by Lamport and Jim Gray in "Consensus on Transaction Commit"
- One instance of the consensus algorithm is used for each Prepared/Aborted decision

Paxos Overview

- Too hard to explain here in detail
- But, essentially the TM functionality is replicated in multiple **acceptors**
  - Algorithm makes progress provided a majority of acceptors are working
  - Messages can be lost, repeated, arrive in an arbitrary order (but can't be tampered with)
  - If you deploy 5 acceptors, you can get a MTTF of about 12 years (assuming a MTTF of 48 hours, and MTTR of 1 hour per acceptor)
  - Goes up to 600 years if you use 7!

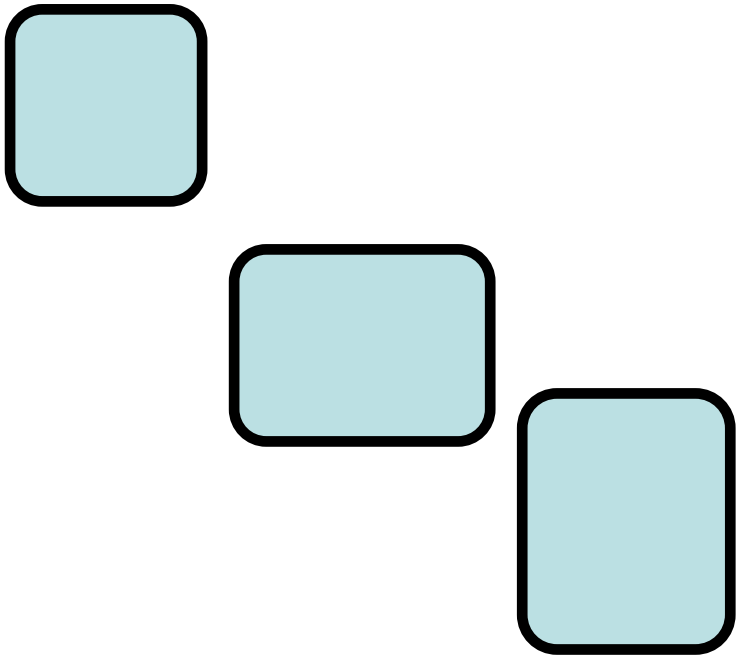

Client
Acceptors
RMs
GetTimetable
Timetable
DoActions
TID
Prepare
Prepared/Aborted
Commit/Abort
GetResult
Result

What are we doing?

- Co-schedule a set of actions
- Actions are:
  - **Make** - create resv.
  - **Modify** - change resources in resv.
  - **Move** - change schedule of resv.
  - **Cancel** - remove resv.
- Can co-schedule any set of these...
- Not all RMs have to support all of these - can just abort with appropriate error

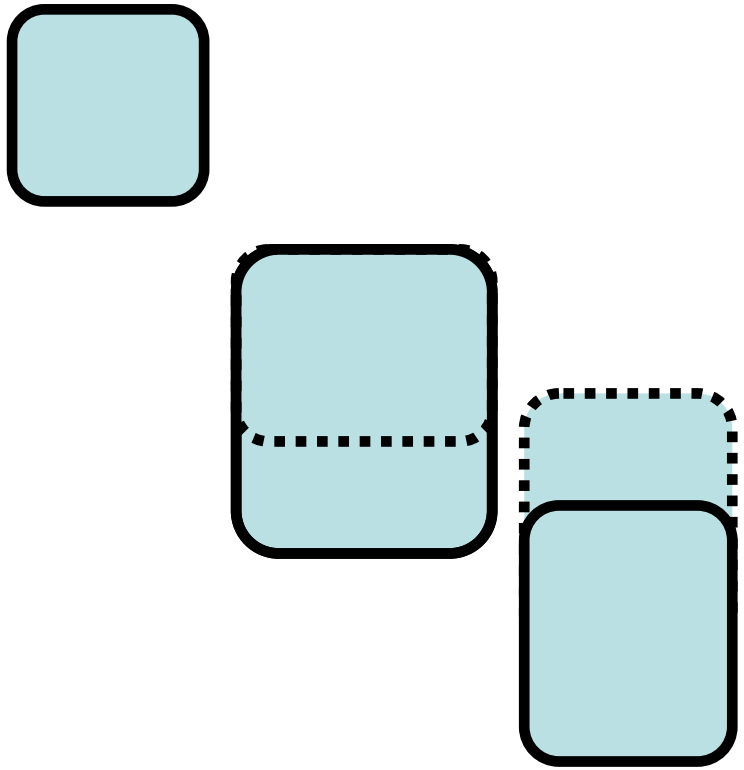
Example

- Co-schedule three **Make**s on X,Y,Z
  - X starts at 11.00, runs for one hour
  - Y starts at 12.15, runs for one hour
  - Z starts at 13.00, runs for 90 mins



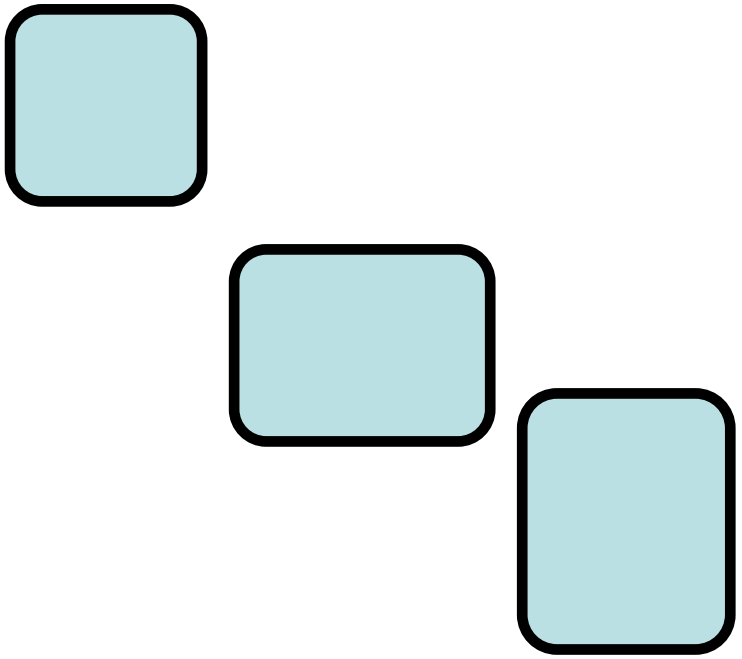
Example

- Need 30 mins longer for Y
- Co-schedule **Modify** on Y with **Move** on Z



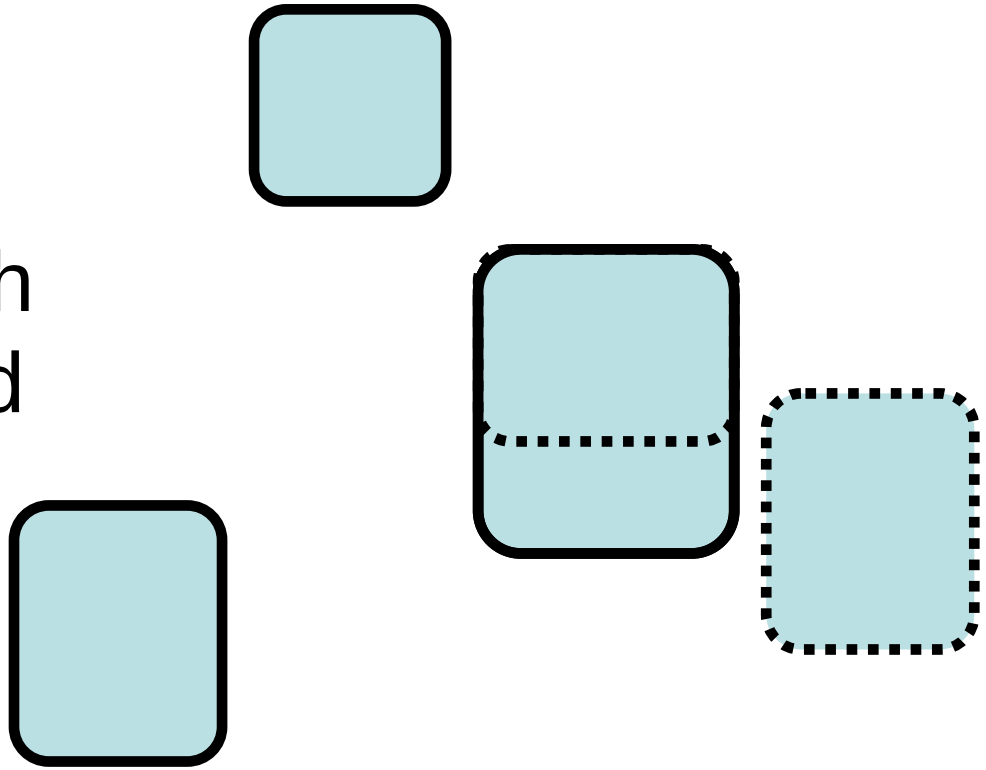
Example

- But this fails, saying that Z doesn't support **Move**
- So we're back here (nothing changed)



Example

- Try again...
- Co-schedule **Modify** on Y with **Cancel** on Z and **Make** on W



Describing the reservation

- It’s all XML. Send a set of actions...
- Make element contains:
  - **Resource** - where
  - **Schedule** - when
  - **Work** - what

- Acceptors look at the “where” part, so they know what to talk to
- Don’t look at Work/Schedule
- Could even be encrypted...

Other messages

- Response for successful **Make** is an **Ident** element

- For **Cancel**, you send a **Resource** element, and the **Ident** element
- For **Move**, you send a **Resource**, the **Ident**, and a new **Schedule**
- For **Modify**, you send a **Resource**, the **Ident** and a new **Work** description

Does it work?

- Yes!
- Have a working implementation
- XML over HTTP (no SOAP)
- Two RMs:
  - PBSPro scheduler
  - Calient DiamondWave network switch
- Co-scheduled 10 compute jobs and 2 switches at iGrid

- It’s available for download! (but...)

What's missing?

- A couple of things not in the first release!
  - Security (but the model is thought out)
  - Writing state to stable storage

Resources

- Everything will happen here!

  http://www.cct.lsu.edu/personal/maclaren/CoSched/

- Page includes:
  - Software for download
  - Mailing list details
- Page **will** include:
  - Documentation!
  - Links to those cool papers