Grid-Aware Evaluation of Regular Path Queries on Spatial Networks

Zhuo Miao, *University of Victoria, Canada*

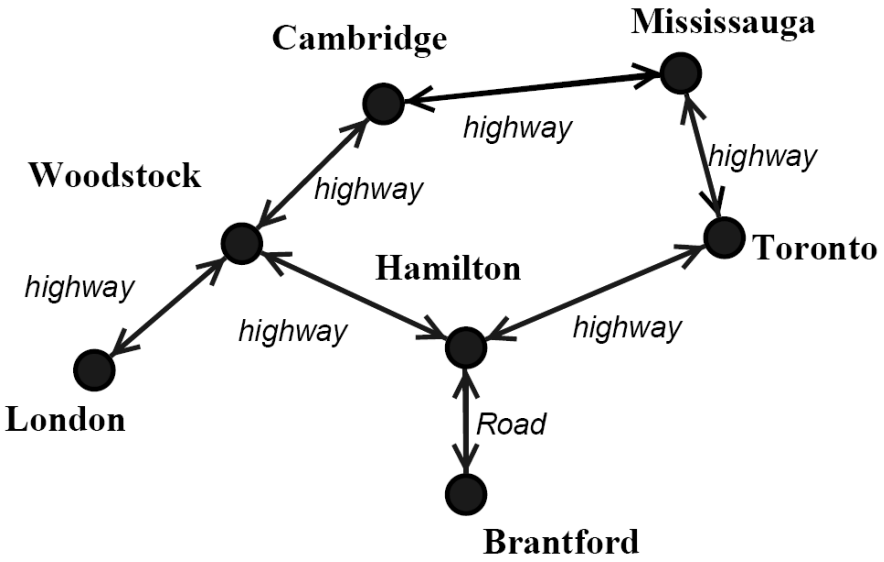
Dan Stefanescu, *Suffolk University, USA*

Alex Thomo, *University of Victoria, Canada*

Spatial Networks

Can be represented by graphs that are:

- Directed
  - There might be one-way streets
- Labeled
  - Highway
  - Road
  - Street
  - Bridge, etc.
- Weighted
  - Kilometric cost
  - Time to traverse, etc.



Typical user demand:

Find shortest paths starting from an origin point.

Regular Path Queries (RPQs)

- Essentially regular expressions over the network alphabet:
  - {highway, road, street, bridge, …}

- Useful for expressing user preferences. **E.g.**

  "I prefer highways and I am willing to tolerate up to one provincial road or city street!"

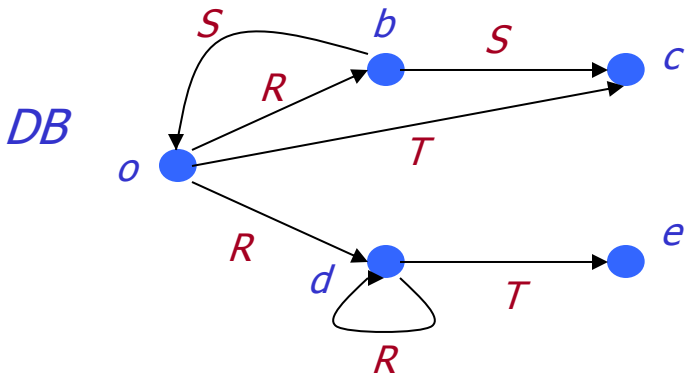
  highway* · (road + street) · highway*

- Now:

  Find shortest paths starting from an origin point, and which *spell some word in the regular query language*.

Databases and Queries

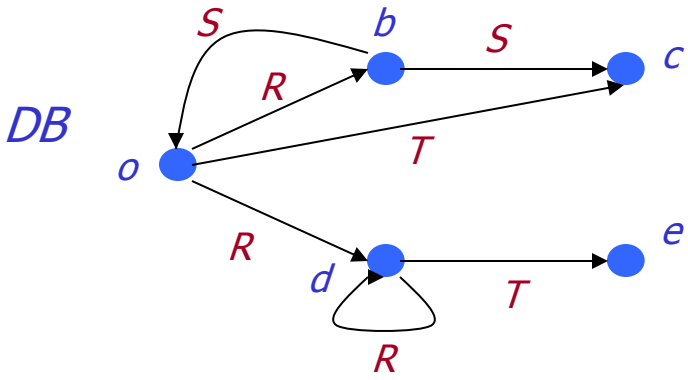
- *DB* is a graph labeled with symbols from Δ

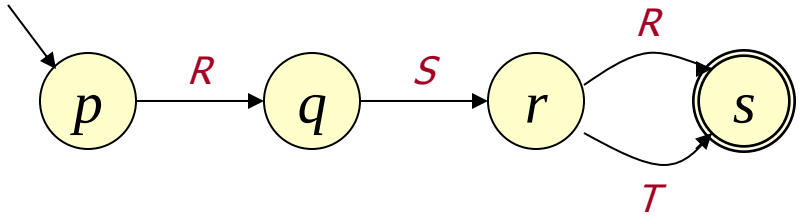


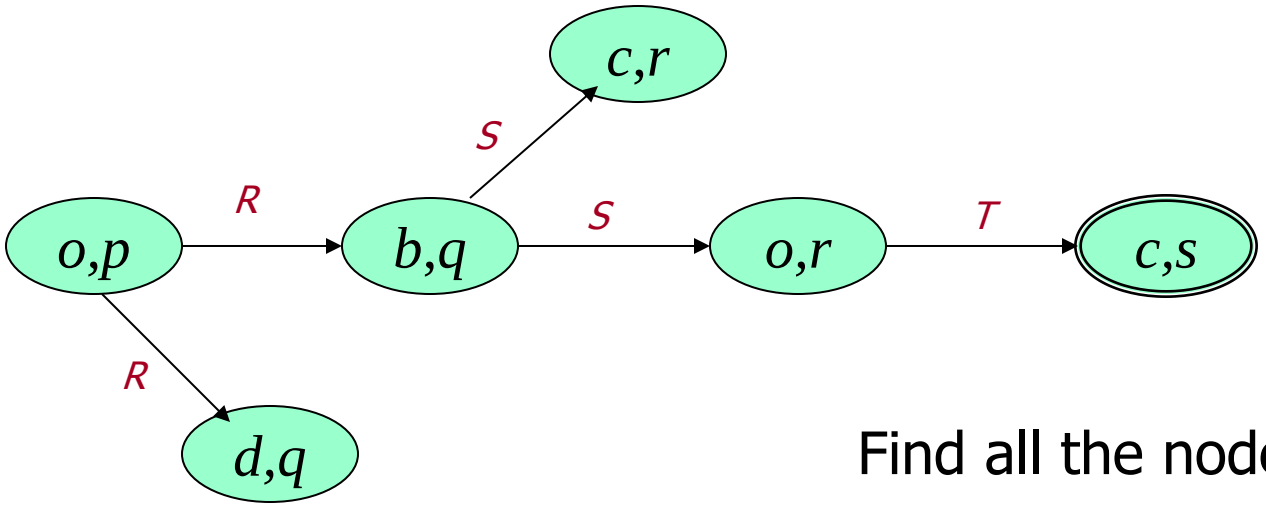
- $Q = RS(R+T)$
  - In general, a regular language on Δ.
- ans(Q,DB) = $\{x : o \overset{w}{\rightsquigarrow} x \in DB,\ w \in Q\}$

  {c} for the above query.

Evaluation of queries







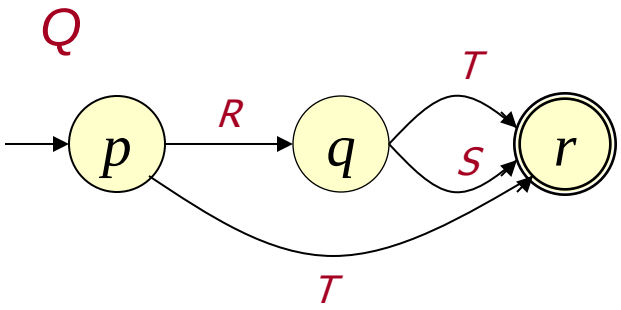
Find all the nodes:

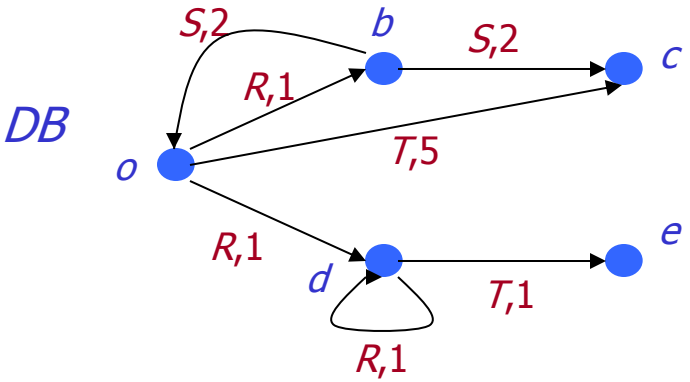
a. reachable from (o,p), and

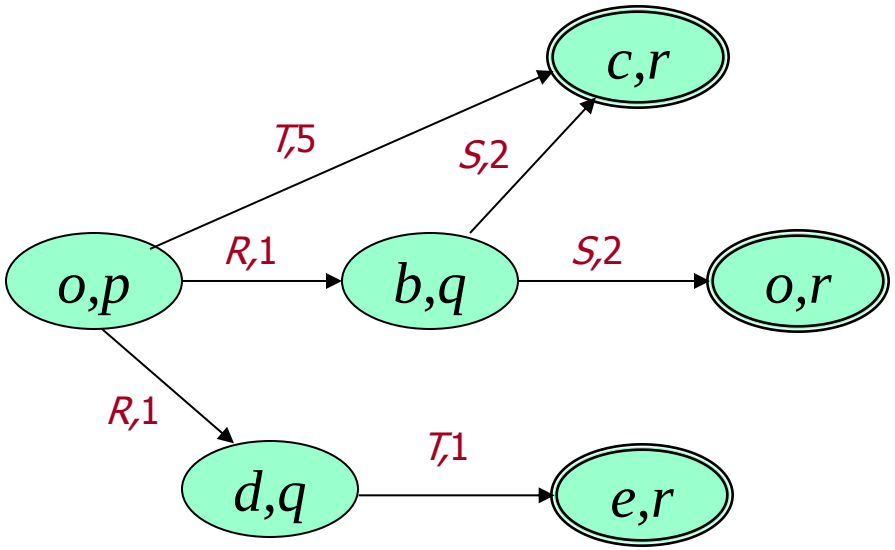
b. associated with a final state

e.g. (c,s)

Weights on Database Edges







Now, apply shortest path algo. on the green graph.

Weighted Answer: {(c,3), (e,r), …}

Evaluation of queries under distortion

- Shortest path algorithms:
  1. Initialize a queue to empty
  2. Insert initial node (o,p_0) in queue.
  3. Loop until queue is empty
     a. remove a node from the queue (according to the queue policy)
     b. "label" with a weight the neighbor nodes (poss. relaxation step)
     c. insert neighbors in queue

What's really inserted in the queue are (node,weight) pairs.

The "labeling" is done by:

if there is an edge a--*k*-->b, and

(a,n) is the pair we removed from queue, and

m is the current weight "label" of b, then

the new "label" m' for b will be: $m' = \min(n+k, m)$

Label setting vs. Label Correcting Algorithms

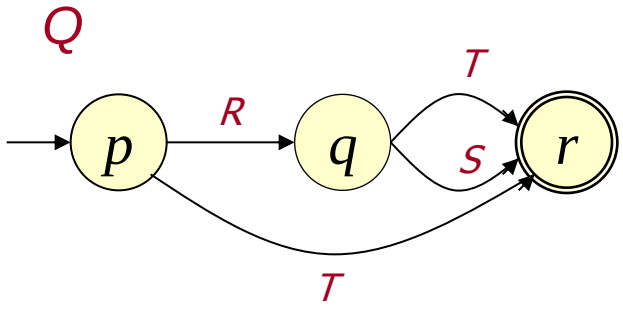
- Label setting algorithms: Use priority queue
  - E.g. Dijkstra's
  - When a node is labeled we know for sure that it's the best label, i.e. it will be the weight of the shortest path from the source to that node.

- Label correcting algorithms:
  - E.g. FIFO, or SLF-LLL queue
  - When a node is labeled we don't know for sure that it's the best label, possibly we will improve it later.
  - Appropriate to extend to distributed setting…

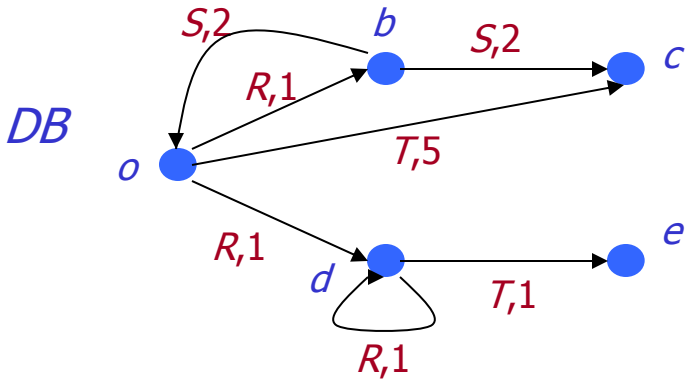
Let's apply it…

We will build the green graph implicitly and lazily.

Along the way we will apply the shortest path general algorithm.



Queue : (o,p,0)

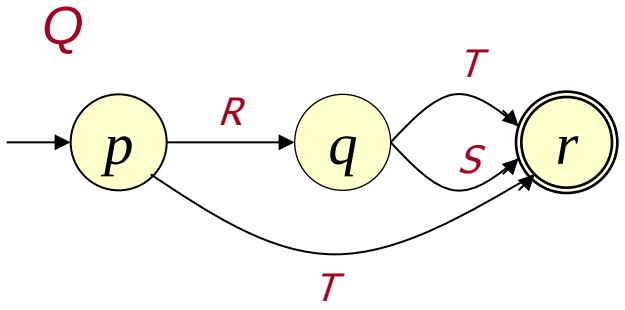


Object-State-Weight Table : Here we store the "labeling"s of the green graph nodes.

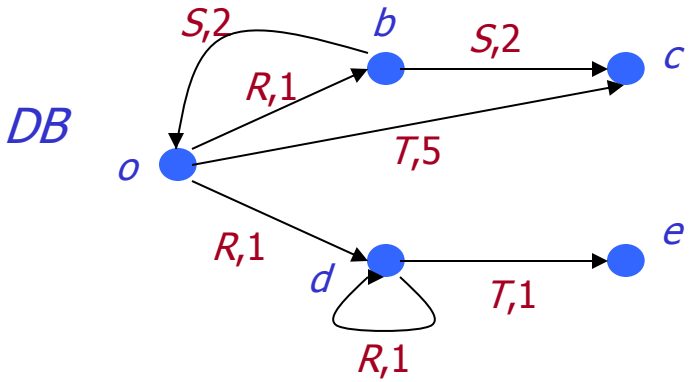
Let's apply it…

We will build the green graph implicitly and lazily.

Along the way we will apply the shortest path general algorithm.



Queue : (c,r,5) (b,q,1) (d,q,1)



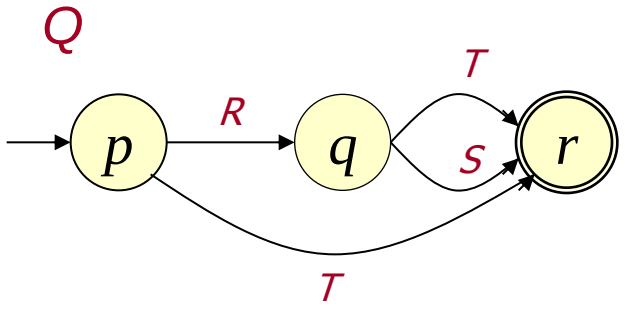
Object-State-Weight Table : Here we store the "labeling"s of the green graph nodes.

(o,p,0)

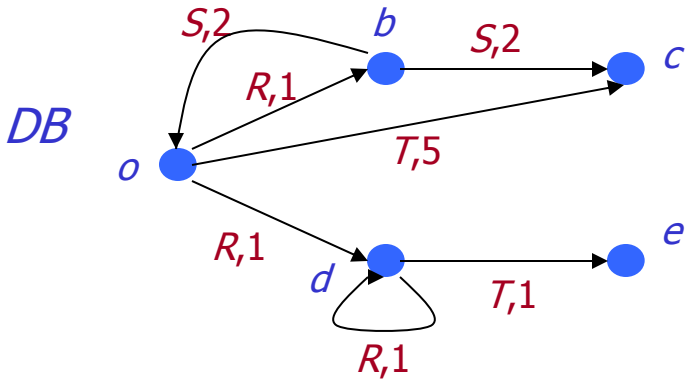
Let's apply it…

We will build the green graph implicitly and lazily.

Along the way we will apply the shortest path general algorithm.



Queue : (b,q,1) (d,q,1)



Object-State-Weight Table : Here we store the "labeling"s of the green graph nodes.

(o,p,0) (c,r,5)

…and so on…

Distributed Algorithm

- The nodes are partitioned in different processors, which don't share memory.
- Communication is achieved through asynchronous message passing.

Idea

- Partition the *Object-State-Weight* table among the processors
- When dequeueing see whether the corresponding object is in the local *Object-State-Weight* table.
  - If yes, proceed as previously
  - Otherwise pack the dequeued triple in a message and send it to the processor holding the real DB node.

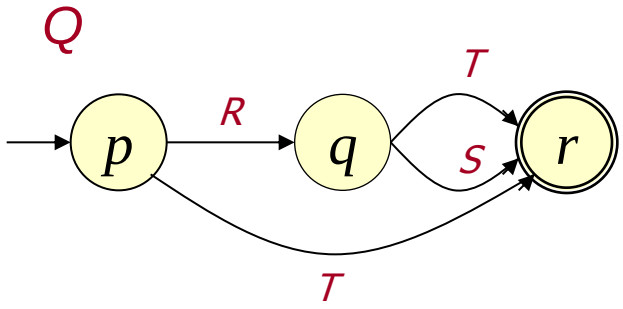
- Terminate when the processing queues of all processors are empty and there is no message sent but not yet received.

Distributed Algorithm

Suppose:

o, d are in processor P1

b, c, e are in processor P2



P1:

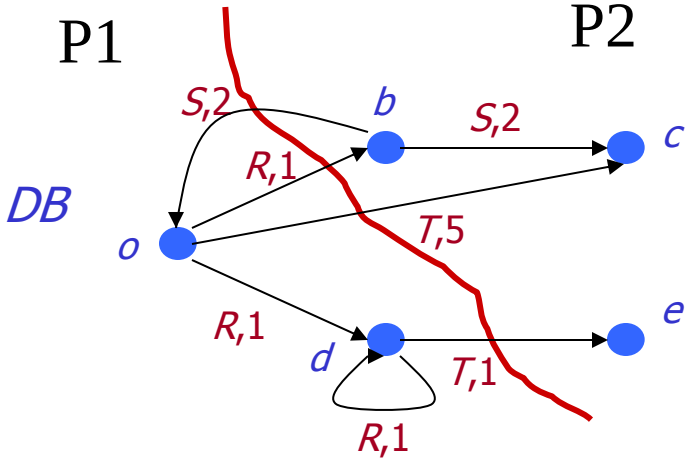
Queue : (o,p,0)

Object-State-Weight Table :

P2:

Queue :

Object-State-Weight Table :

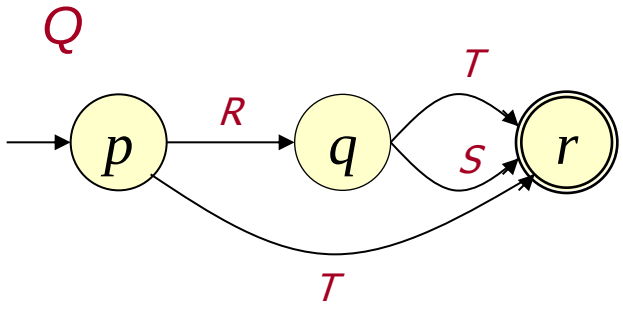


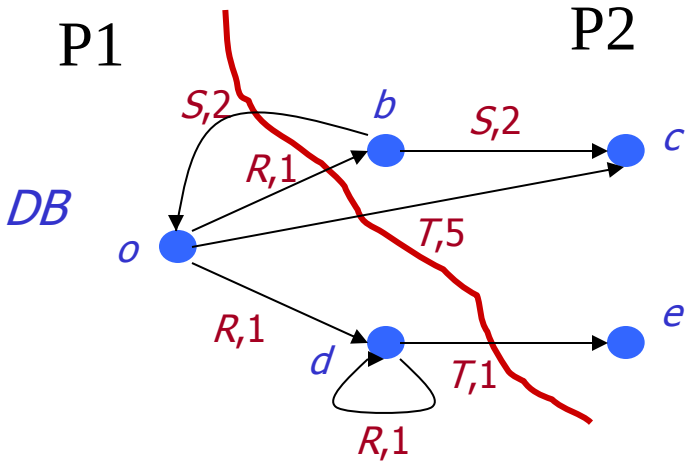
Distributed Algorithm

Suppose:

o, d are in processor P1

b, c, e are in processor P2





P1:

Queue : (c,r,5) (b,q,1) (d,q,1)

Object-State-Weight Table :

(o,p,0)

message (c,r,5)

P2:

Queue :

Object-State-Weight Table :

Challenges in a Grid

- A computational grid is a community of machines
  - primarily destined for other tasks different from the tasks they can be assigned in a grid.

- Machines are allowed to maintain a degree of freedom regarding
  - work they accept and
  - reliability that they offer.

Thus,

- we need to limit the computational stress on the machines, and
- be resilient against machine losses.

Reducing Computational Stress

- One element to investigate in the quest for “balanced” computation is the choice of the data partitioning.
  - It turns out that a good partitioning is interrelated with the database storage scheme.

- We use a clustering RTree (spatial) index.
- With such an index, we not only cluster together (in disk-blocks) edges that are spatially close to each other, but also partition the data among participating grid machines.
  - Namely, each machine locally stores and works on a subset of the RTree leaves.
  - After this RTree partitioning, each machine builds a local RTree index on the blocks assigned to it.

Machine Loss and Termination Detection

We adapt the Dijkstra-Scholten termination detection algorithm.

- Organize active machines in a spanning tree rooted at query originator.
- A (previously passive) machine joins the tree upon the receipt of its first message/task from an active machine which becomes its parent.
- Each active machine uses a local variable “tasks” to keep count of its unacknowledged messages/tasks.
- Messages are acknowledged immediately unless they are the “engaging messages,” i.e. messages that result in passive machines becoming active.
- A non-originating active machine can become passive, and attain “local termination” if its local processing queue is empty and it has no unacknowledged messages.
  - At that time, the respective machine acknowledges its parent and severs its connection from the spanning tree.
- The processing of the query ends when the originating machine has no unacknowledged messages left.

Machine Loss and Termination Detection

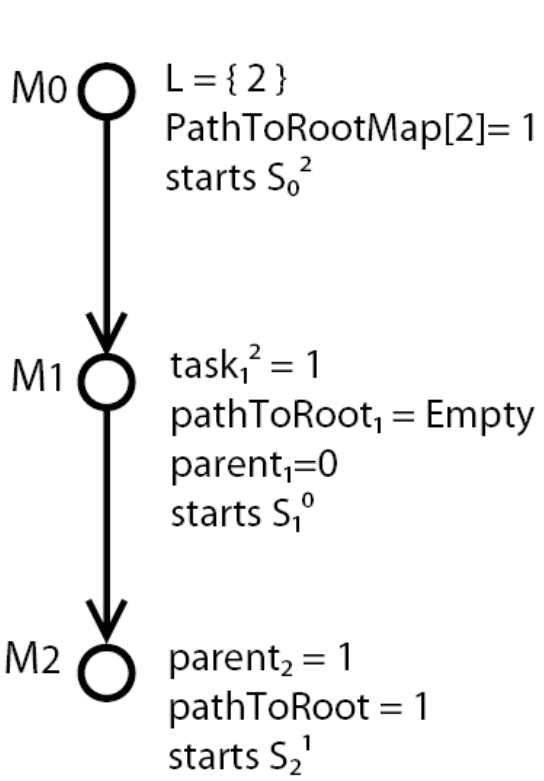
In a grid:

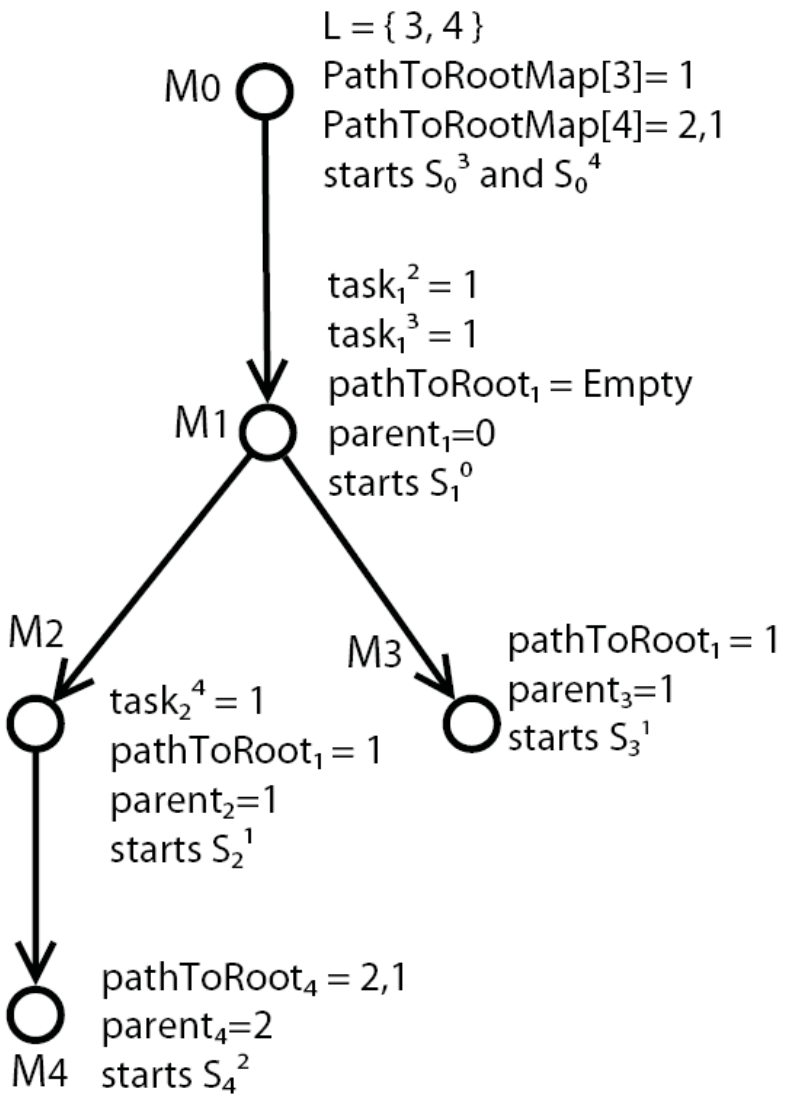
- Practical assumption: M_0, the originating user machine, does not fault.

- Each node in the spanning tree is responsible for monitoring the health of its parent.
- Each leaf node is monitored by a “dummy offspring leaf.”
  - We let the root machine M_0 to play this (additional) role!

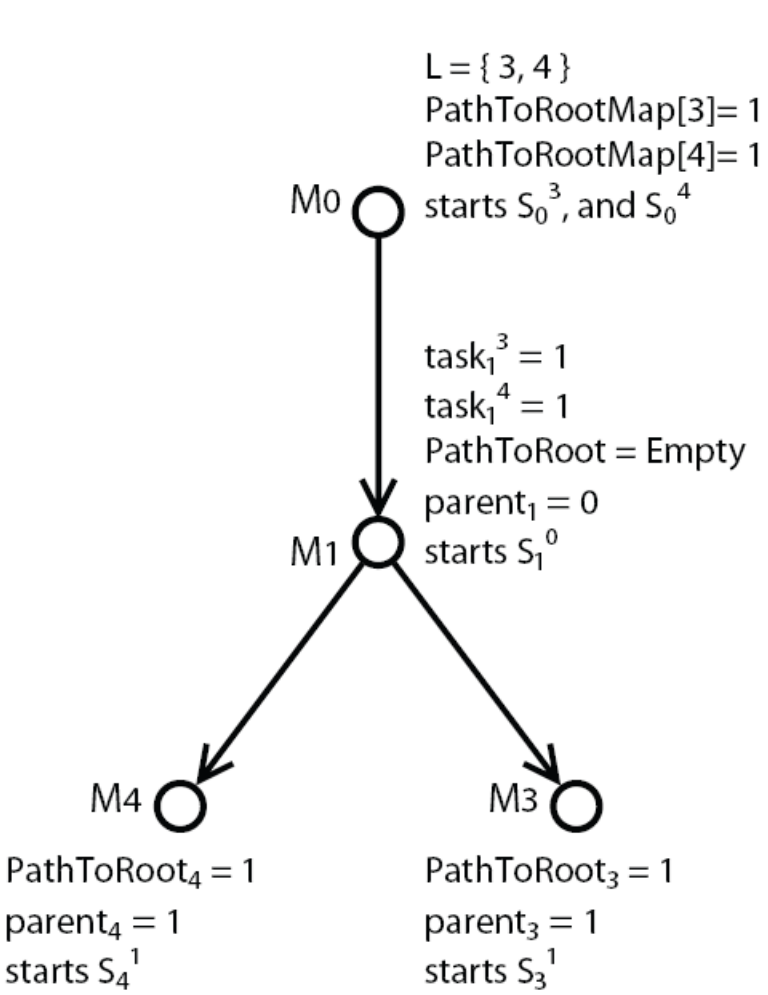
- As machines fault, the spanning tree needs to be rebuilt on the fly.
  - Live spanning tree orphans need to acquire new parents, and
  - All nodes need to readjust their termination bookkeeping in order to account for faulty machines.
- Upon detecting its parent loss, a nonroot node makes, as the new parent, the closest alive ancestor in the spanning tree.

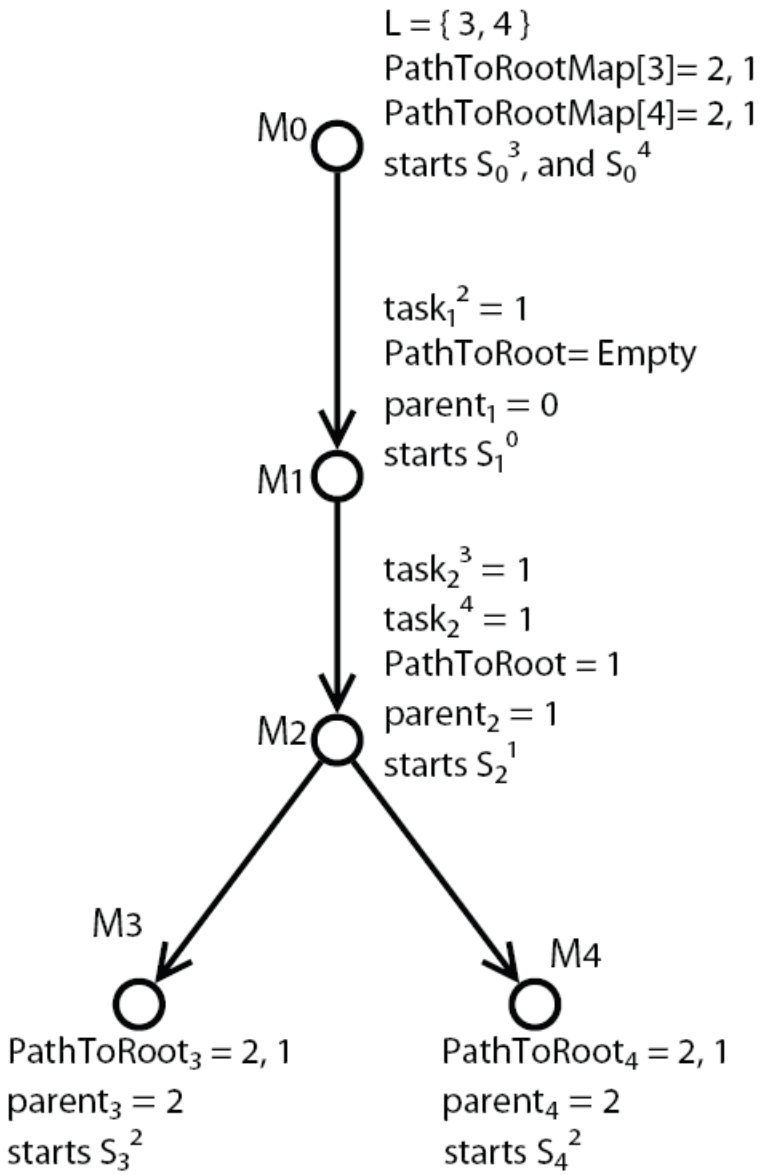
Example


M0
L = { 2 }
PathToRootMap[2]= 1
starts S_0^2
M1
$task_1^2 = 1$
$pathToRoot_1$ = Empty
$parent_1$=0
starts S_1^0
M2
$parent_2 = 1$
pathToRoot = 1
starts S_2^1

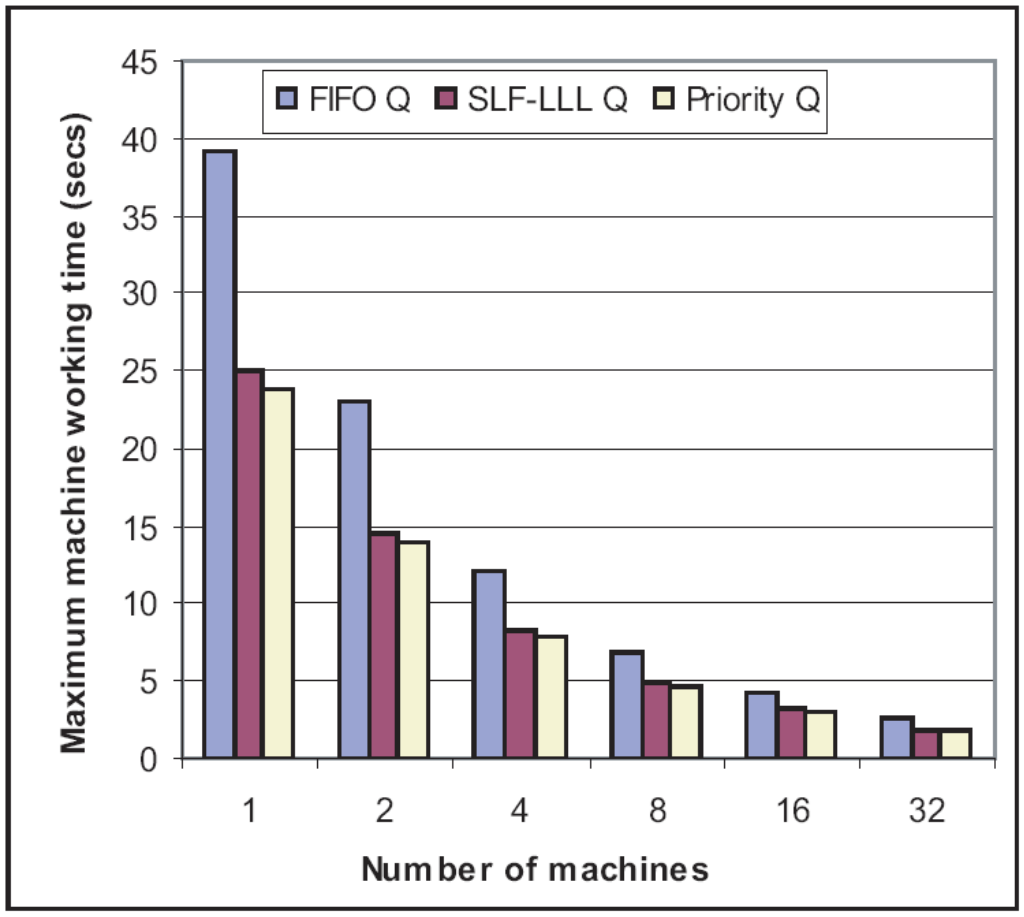

M0
L = { 3, 4 }
PathToRootMap[3]= 1
PathToRootMap[4]= 2,1
starts S_0^3 and S_0^4
M1
$task_1^2 = 1$
$task_1^3 = 1$
$pathToRoot_1$ = Empty
$parent_1$=0
starts S_1^0
M2
$task_2^4 = 1$
$pathToRoot_1 = 1$
$parent_2$=1
starts S_2^1
M3
$pathToRoot_1 = 1$
$parent_3$=1
starts S_3^1
M4
$pathToRoot_4$ = 2,1
$parent_4$=2
starts S_4^2

Example


L = { 3, 4 }
PathToRootMap[3]= 1
PathToRootMap[4]= 1
M0
starts S_0^3, and S_0^4
$task_1^3 = 1$
$task_1^4 = 1$
PathToRoot = Empty
$parent_1 = 0$
M1
starts S_1^0
M4
M3
$PathToRoot_4 = 1$
$parent_4 = 1$
starts S_4^1
$PathToRoot_3 = 1$
$parent_3 = 1$
starts S_3^1


L = { 3, 4 }
PathToRootMap[3]= 2, 1
PathToRootMap[4]= 2, 1
M0
starts S_0^3, and S_0^4
$task_1^2 = 1$
PathToRoot= Empty
$parent_1 = 0$
M1
starts S_1^0
$task_2^3 = 1$
$task_2^4 = 1$
PathToRoot = 1
$parent_2 = 1$
M2
starts S_2^1
M3
M4
$PathToRoot_3 = 2, 1$
$parent_3 = 2$
starts S_3^2
$PathToRoot_4 = 2, 1$
$parent_4 = 2$
starts S_4^2

Stress Reduction on Machines



Best use Priority Queue

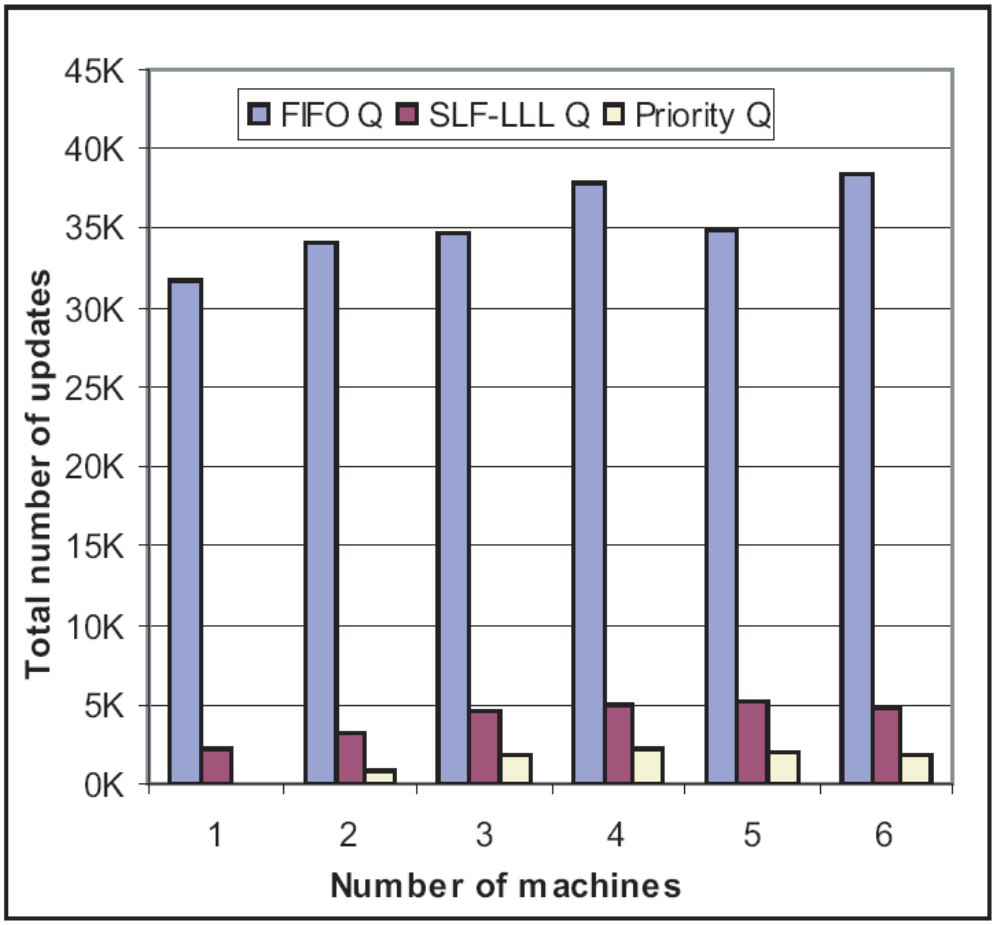
Reduction of stress is (almost) half as the number of machines doubles at each point.

Also, we observe

that the use of SLF-LLL or just FIFO queues, in the name of reducing the computational overhead of maintaining a

priority queue, is in fact not justified.

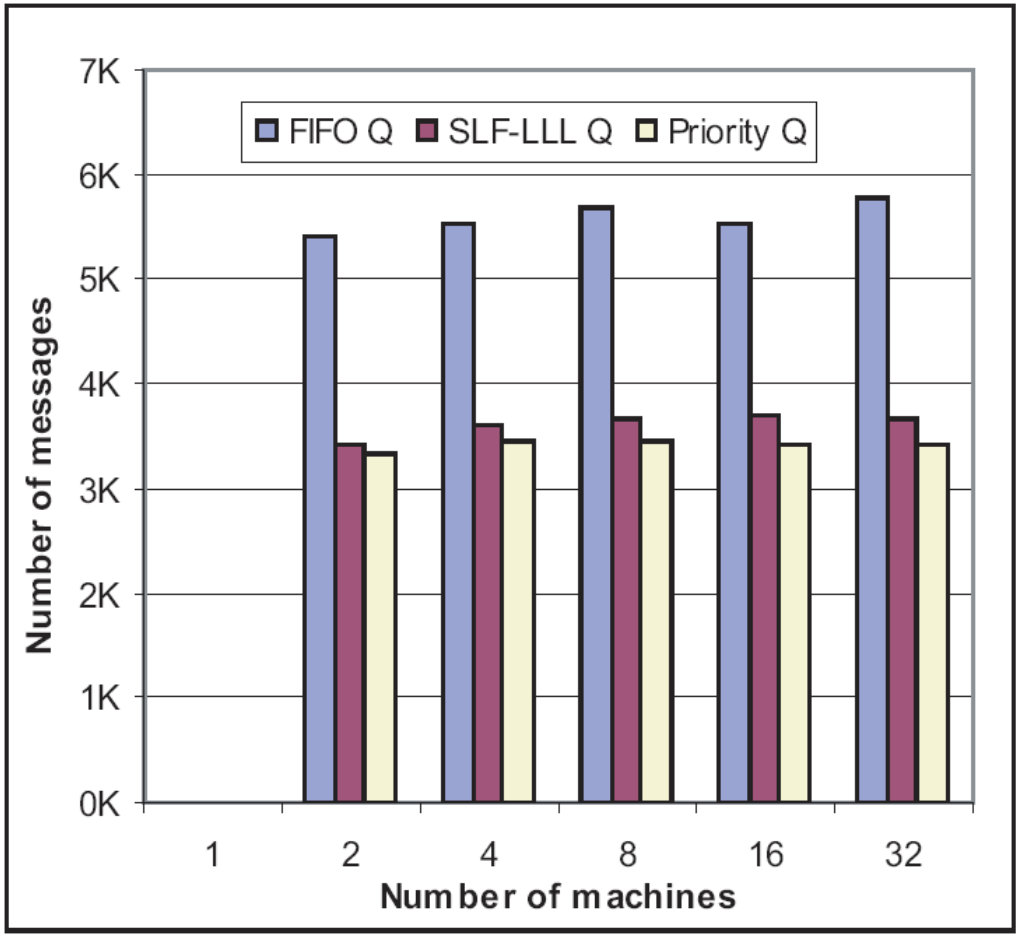
Quality of Intermediate Answers



Best use Priority Queue:

Less updates means less inconsistent query answers that get their weight corrected.

Number of Messages



Best use Priority Queue:

Our algorithm is *not* message intensive.

When using priority queues, the number of messages is approximately 3500 and this is quite negligible for today's high speed networks.

Questions?

References

- Zhuo Miao, Dan C. Stefanescu, Alex Thomo. Grid-Aware Evaluation of Regular Path Queries on Spatial Networks. AINA 2007: 158-165
- Maryam Shoaran, Alex Thomo. Distributed Multi-source Regular Path Queries. ISPA Workshops 2007: 365-374
- Gösta Grahne, Alex Thomo, William W. Wadge. Preferentially Annotated Regular Path Queries. ICDT 2007: 314-328
- Gösta Grahne, Alex Thomo. Regular path queries under approximate semantics. Ann. Math. Artif. Intell. 46(1-2): 165-190 (2006)
- Dan C. Stefanescu, Alex Thomo. Enhanced Regular Path Queries on Semistructured Databases. EDBT Workshops 2006: 700-711.
- Dan C. Stefanescu, Alex Thomo, Lida Thomo. Distributed evaluation of generalized path queries. SAC 2005: 610-616