Static Scheduling for Large-Scale Heterogeneous Platforms

Yves Robert
École Normale Supérieure de Lyon

joint work with
Larry Carter, Henri Casanova, Jeanne Ferrante, Yang Yang
Olivier Beaumont, Arnaud Legrand, Loris Marchal, Frédéric Vivien

Yves.Robert@ens-lyon.fr
http://graal.ens-lyon.fr/~yrobert

ΙΠΔΠΣ'2006

Evolution of parallel machines

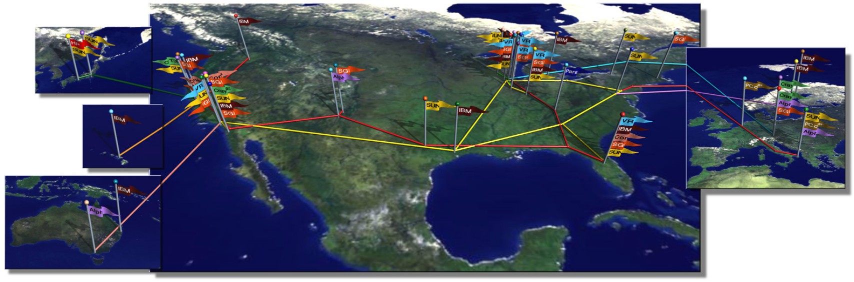
From good old parallel architectures . . .

Evolution of parallel machines

. . . to heterogeneous clusters . . .

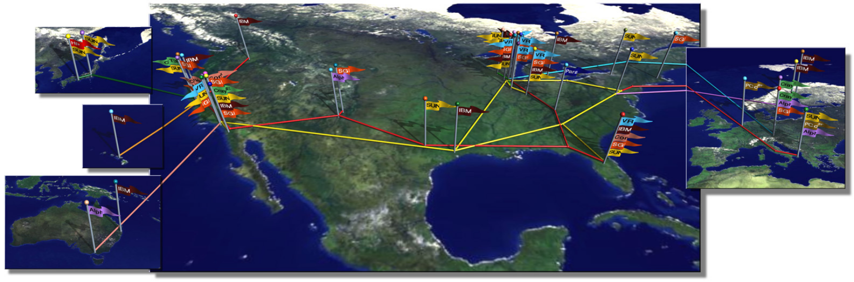
Evolution of parallel machines

. . . and soon to the Holy Grid?



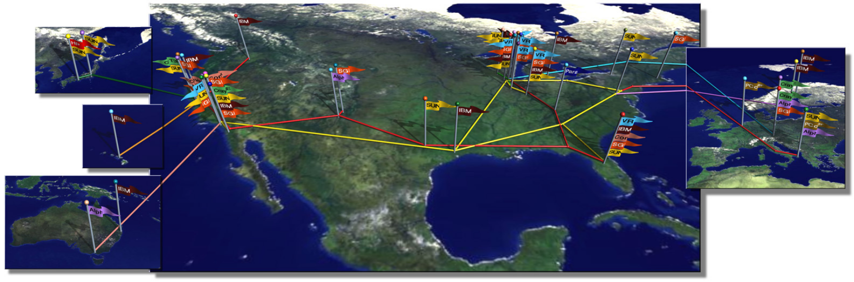
Evolution of parallel machines

. . . and soon to the Holy Grid?



Parallel algorithm design and scheduling were already difficult tasks with homogeneous machines

. . . and soon to the Holy Grid?



Parallel algorithm design and scheduling were already difficult tasks with homogeneous machines

On heterogeneous platforms, it gets worse

Target platforms: Large-scale heterogenous platforms
(networks of workstations, clusters, collections of clusters, grids, ...)

New problems

- Heterogeneity of processors (CPU power, memory)
- Heterogeneity of communication links
- Irregularity of interconnection networks
- Non-dedicated platforms

Need to adapt algorithms and scheduling strategies: new objective functions, new models

New platforms, new problems, **new solutions**

Target platforms: Large-scale heterogenous platforms
(networks of workstations, clusters, collections of clusters, grids, ...)

New problems

- Heterogeneity of processors (CPU power, memory)
- Heterogeneity of communication links
- Irregularity of interconnection networks
- Non-dedicated platforms

Need to adapt algorithms and scheduling strategies: new objective functions, new models

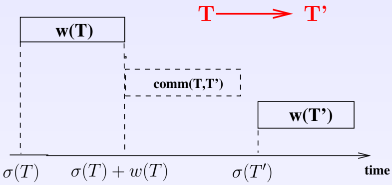
Outline

Outline

Traditional scheduling – Framework

- Application = DAG $G = (\mathcal{T}, E, w)$
  - $\mathcal{T}$ = set of tasks
  - E = dependence constraints
  - $w(T)$ = computational cost of task T (execution time)
  - $c(T, T')$ = communication cost (data sent from T to T')
- Platform
  - Set of p identical processors
- Schedule
  - $\sigma(T)$ = date to begin execution of task T
  - alloc(T) = processor assigned to it

Traditional scheduling – Constraints



- Data dependences If $(T,T') \in E$ then
  - if alloc(T) = alloc(T') then $\sigma(T)+w(T) \leq \sigma(T')$
  - if alloc$(T) \neq$ alloc(T') then $\sigma(T)+w(T)+c(T,T') \leq \sigma(T')$
- Resource constraints

$$\text{alloc}(T) = \text{alloc}(T') \Rightarrow$$
$$(\sigma(T)+w(T) \leq \sigma(T')) \text{ or } (\sigma(T')+w(T') \leq \sigma(T))$$

Traditional scheduling – Objective functions

- Makespan or total execution time

$$MS(\sigma) = \max_{T \in \mathcal{T}} (\sigma(T) + w(T))$$

- Other classical objectives:
  - Sum of completion times
  - With release dates: maximum flow (response time), or sum flow
  - Fairness oriented: maximum stretch, or sum stretch

Traditional scheduling – About the model

- Simple but OK for computational resources
  - No CPU sharing, even in models with preemption
  - At most one task running per processor at any time-step
- Very crude for network resources
  - Unlimited number of simultaneous sends/receives per processor
  - No contention $\rightarrow$ unbounded bandwidth on any link
  - Fully connected interconnection graph (clique)
- In fact, model assumes infinite network capacity

Makespan minimization

- NP-hardness
  - $Pb(p)$ NP-complete for independent tasks and no communications ($E = \emptyset$, $p = 2$ and $c = \overline{0}$)
  - $Pb(p)$ NP-complete for UET-UCT graphs ($w = c = \overline{1}$)

- Approximation algorithms
  - Without communications, list scheduling is a $(2 - \frac{1}{p})$-approximation
  - With communications, result extends to coarse-grain graphs
  - With communications, no λ-approximation in general

List scheduling – Without communications

Initialization:

1. Compute priority level of all tasks
2. Priority queue $=$ list of free tasks (tasks without predecessors) sorted by priority

While there remain tasks to execute:

1. Add new free tasks, if any, to the queue.
2. If there are q available processors and r tasks in the queue, remove first $\min(q, r)$ tasks from the queue and execute them

Priority level

- Use critical path: longest path from the task to an exit node
- Computed recursively by a bottom-up traversal of the graph

List scheduling – Without communications

Initialization:

1. Compute priority level of all tasks
2. Priority queue = list of free tasks (tasks without predecessors) sorted by priority

While there remain tasks to execute:

1. Add new free tasks, if any, to the queue.
2. If there are q available processors and r tasks in the queue, remove first $\min(q, r)$ tasks from the queue and execute them

Priority level

- Use critical path: longest path from the task to an exit node
- Computed recursively by a bottom-up traversal of the graph

List scheduling – With communications (1/2)

- Priority level
  - Use *pessimistic* critical path: include all edge costs in the weight
  - Computed recursively by a bottom-up traversal of the graph

- MCP *Modified Critical Path*
  - Assign free task with highest priority to *best* processor
  - Best processor = finishes execution first, given already taken scheduling decisions
  - Free tasks may not be ready for execution (communication delays)
  - May explore inserting the task in empty slots of schedule
  - Complexity $O(|V| \log |V| + (|E| + |V|)p)$

List scheduling – With communications (2/2)

- EFT *Earliest Finish Time*
  - Dynamically recompute priorities of free tasks
  - Select free task that finishes execution first (on best processor), given already taken scheduling decisions
  - Higher complexity $O(|V|^3 p)$
  - May miss "urgent" tasks on critical path
- Other approaches
  - Two-step: clustering + load balancing
    - DSC Dominant Sequence Clustering $O((|V| + |E|) \log |V|)$
    - LLB List-based Load Balancing $O(C \log C + |V|)$ (C number of clusters generated by DSC)
  - Low-cost: FCP Fast Critical Path
    - Maintain constant-size sorted list of free tasks:
    - Best processor = first idle or the one sending last message
    - Low complexity $O(|V| \log p + |E|)$

Extending the model to heterogeneous clusters

- Task graph with n tasks $T_1, \ldots, T_n$.
- Platform with p heterogeneous processors $P_1, \ldots, P_p$.
- Computation costs:
  - w_{iq} = execution time of T_i on P_q
  - $\overline{w_i} = \frac{\sum_{q=1}^{p} w_{iq}}{p}$ average execution time of T_i
  - particular case: consistent tasks $w_{iq} = w_i \times \gamma_q$
- Communication costs:
  - data(i,j): data volume for edge $e_{ij} : T_i \rightarrow T_j$
  - v_{qr}: communication time for unit-size message from P_q to P_r (zero if $q = r$)
  - com(i,j,q,r) = data$(i,j) \times v_{qr}$ communication time from T_i executed on P_q to P_j executed on P_r
  - $\overline{\text{com}_{ij}}$ = data$(i,j) \times \frac{\sum_{1 \leq q,r \leq p, q \neq r} v_{qr}}{p(p-1)}$ average communication cost for edge $e_{ij} : T_i \rightarrow T_j$

Rewriting constraints

Dependences For $e_{ij} : T_i \rightarrow T_j$, $q = \text{alloc}(T_i)$ and $r = \text{alloc}(T_j)$:

$$\sigma(T_i) + w_{iq} + \text{com}(i, j, q, r) \leq \sigma(T_j)$$

Resources If $q = \text{alloc}(T_i) = \text{alloc}(T_j)$, then

$$(\sigma(T_i) + w_{iq} \leq \sigma(T_j)) \text{ or } (\sigma(T_j) + w_{jq} \leq \sigma(T_i))$$

Makespan

$$\max_{1 \leq i \leq n} \left(\sigma(T_i) + w_{i,\text{alloc}(T_i)}\right)$$

HEFT: Heterogeneous Earliest Finish Time

1. Priority level:
   - $\text{rank}(T_i) = \overline{w_i} + \max_{T_j \in \text{Succ}(T_i)} (\overline{\text{com}_{ij}} + \text{rank}(T_j))$,
     where $\text{Succ}(T)$ is the set of successors of T
   - Recursive computation by bottom-up traversal of the graph
2. Allocation
   - For current task T_i, determine best processor P_q: minimize $\sigma(T_i) + w_{iq}$
   - Enforce constraints related to communication costs
   - Insertion scheduling: look for $t = \sigma(T_i)$ s.t. P_q is available during interval $[t, t + w_{iq}[$
3. Complexity: same as MCP without/with insertion

Bibliography – Traditional scheduling

- Introductory book:
  Distributed and parallel computing, H. El-Rewini and T. G. Lewis, Manning 1997
- FCP:
  On the complexity of list scheduling algorithms for distributed-memory systems, A. Radulescu and A.J.C. van Gemund, 13th ACM Int Conf. Supercomputing (1999), 68-75
- HEFT:
  Performance-effective and low-complexity task scheduling for heterogeneous computing, H. Topcuoglu and S. Hariri and M.-Y. Wu, IEEE TPDS 13, 3 (2002), 260-274

What's wrong?

- ☺ Nothing (still may need to map a DAG onto a platform!)
- ☹ Absurd communication model:
  complicated: many parameters to instantiate
  while not realistic (clique + no contention)
- ☹ Wrong metric: need to relax makespan minimization objective

What's wrong?

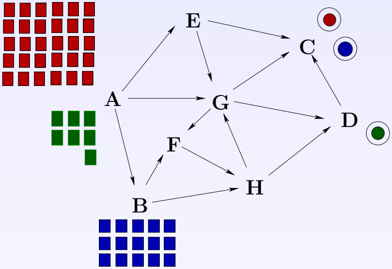
- ☺ Nothing (still may need to map a DAG onto a platform!)
- ☹ Absurd communication model:
  complicated: many parameters to instantiate
  while not realistic (clique + no contention)
- ☹ Wrong metric: need to relax makespan minimization objective

What's wrong?

- ☺ Nothing (still may need to map a DAG onto a platform!)
- ☹ Absurd communication model:
  complicated: many parameters to instantiate
  while not realistic (clique + no contention)
- ☹ Wrong metric: need to relax makespan minimization objective

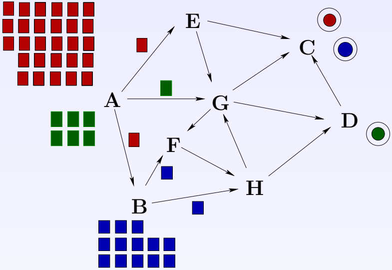
Outline

Problem



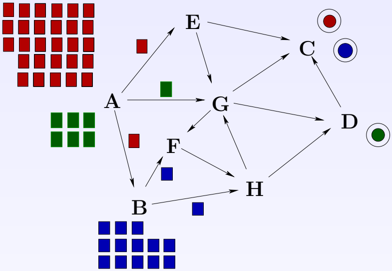
- Routing sets of messages from sources to destinations
- Paths not fixed a priori
- Packets of same message may follow different paths

Hypotheses



- A packet crosses an edge within one time-step
- At any time-step, at most one packet crosses an edge

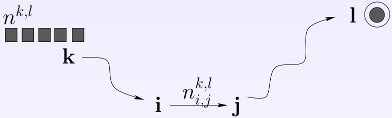
Hypotheses



- A packet crosses an edge within one time-step
- At any time-step, at most one packet crosses an edge

Scheduling: for each time-step, decide which packet crosses any given edge

Notation



- $n^{k,l}$: total number of packets to be routed from k to l
- $n_{i,j}^{k,l}$: total number of packets routed from k to l and crossing edge (i, j)

Lower bound

Congestion $C_{i,j}$ of edge (i,j)
$=$ total number of packets that cross (i,j)

$$C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l} \qquad C_{\max} = \max_{i,j} C_{i,j}$$

$C_{\max}$ lower bound on schedule makespan

$$C^* \geq C_{\max}$$

$\Rightarrow$ "Fluidified" solution in $C_{\max}$?

Lower bound

Congestion $C_{i,j}$ of edge (i,j)
$=$ total number of packets that cross (i,j)

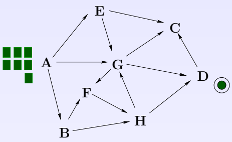
$$C_{i,j} = \sum_{(k,l) \mid n^{k,l} > 0} n_{i,j}^{k,l} \qquad C_{\max} = \max_{i,j} C_{i,j}$$

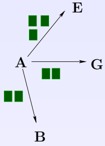
$C_{\max}$ lower bound on schedule makespan

$$C^* \geq C_{\max}$$

$\Rightarrow$ "Fluidified" solution in $C_{\max}$?

Equations (1/2)

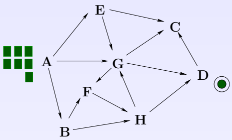


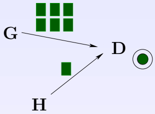


1. Initialization (packets leave node k): $$\sum_{j|(k,j)\in A} n_{k,j}^{k,l} = n^{k,l}$$
2. Reception (packets reach node l): $$\sum_{i|(i,l)\in A} n_{i,l}^{k,l} = n^{k,l}$$
3. Conservation law (crossing intermediate node i):

$$\sum_{i|(i,j)\in A} n_{i,j}^{k,l} = \sum_{i|(j,i)\in A} n_{j,i}^{k,l} \quad \forall (k,l), j \neq k, j \neq l$$

Equations (1/2)

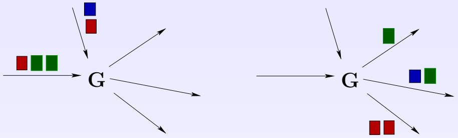




1. Initialization (packets leave node k): $$\sum_{j|(k,j)\in A} n_{k,j}^{k,l} = n^{k,l}$$
2. Reception (packets reach node l): $$\sum_{i|(i,l)\in A} n_{i,l}^{k,l} = n^{k,l}$$
3. Conservation law (crossing intermediate node i):

$$\sum_{i|(i,j)\in A} n_{i,j}^{k,l} = \sum_{i|(j,i)\in A} n_{j,i}^{k,l} \quad \forall (k,l), j \neq k, j \neq l$$

Equations (1/2)



1. Initialization (packets leave node k): $\displaystyle\sum_{j|(k,j)\in A} n_{k,j}^{k,l} = n^{k,l}$
2. Reception (packets reach node l): $\displaystyle\sum_{i|(i,l)\in A} n_{i,l}^{k,l} = n^{k,l}$
3. Conservation law (crossing intermediate node i):

$$\sum_{i|(i,j)\in A} n_{i,j}^{k,l} = \sum_{i|(j,i)\in A} n_{j,i}^{k,l} \quad \forall (k,l), j \neq k, j \neq l$$

Equations (2/2)

4. Congestion

$$C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l}$$

5. Objective function

$$C_{\max} \geq C_{i,j}, \qquad \forall i, j$$

Minimize $C_{\max}$

Linear program in rational numbers: polynomial-time solution. In practice use Maple or Mupad

Equations (2/2)

4. Congestion

$$C_{i,j} = \sum_{(k,l) | n^{k,l} > 0} n_{i,j}^{k,l}$$

5. Objective function

$$C_{\max} \geq C_{i,j}, \qquad \forall i,j$$

Minimize $C_{\max}$

Linear program in rational numbers: polynomial-time solution. In practice use Maple or Mupad

Routing algorithm

1. Compute optimal solution $C_{\max}$, $n_{i,j}^{k,l}$ of previous linear program
2. Periodic schedule:
   - Define $\Omega = \sqrt{C_{\max}}$
   - Use $\left\lceil \frac{C_{\max}}{\Omega} \right\rceil$ periods of length Ω
   - During each period, edge (i,j) forwards (at most)

$$m_{i,j}^{k,l} = \left\lfloor \frac{n_{i,j}^{k,l}\Omega}{C_{\max}} \right\rfloor$$

   packets that go from k to l
3. Clean-up: sequentially process residual packets inside network

Performance

- Schedule is feasible
- Schedule is asymptotically optimal:

$$C_{\max} \leq C^* \leq C_{\max} + O(\sqrt{C_{\max}})$$

Why does it work?

- Relaxation of objective function
- Rational number of packets in LP formulation
- Periods long enough so that rounding down to integer numbers has negligible impact
- Periods numerous enough so that loss in first and last periods has negligible impact
- Periodic schedule, described in compact form

Bibliography – Packet routing

- Survey of results:
  Introduction to parallel algorithms and architectures: arrays, trees, hypercubes, F.T. Leighton, Morgan Kaufmann (1992)
- NP-completeness, approximation algorithm:
  A constant-factor approximation algorithm for packet routing and balancing local vs. global criteria, A. Srinivasan, C.-P. Teo, SIAM J. Comput. 30, 6 (2000), 2051-2068
- Steady-state:
  Asymptotically optimal algorithms for job shop scheduling and packet routing, D. Bertsimas and D. Gamarnik, Journal of Algorithms 33, 2 (1999), 296-318

Outline

Master-worker tasking: framework

Heterogeneous resources

- Processors of different speeds
- Communication links with various bandwidths

Large number of independent tasks to process

- Tasks are atomic
- Tasks have same size

Single data repository

- One master initially holds data for all tasks
- Several workers arranged along a star, a tree or a general graph

Application examples

- Monte Carlo methods
- SETI@home
- Factoring large numbers
- Searching for Mersenne primes
- Particle detection at CERN (LHC@home)
- ... and many others: see BOINC at http://boinc.berkeley.edu

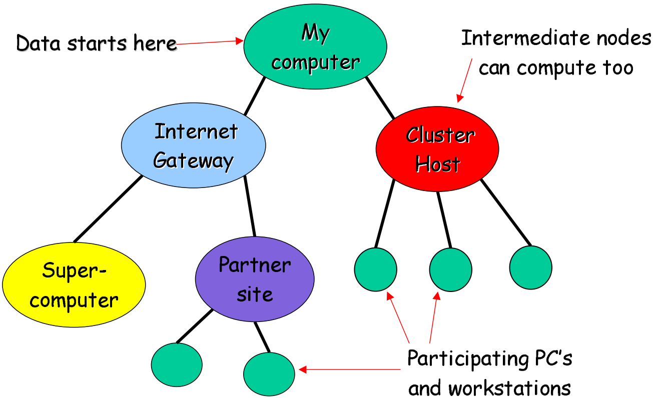
Makespan vs. steady state

Two-different problems

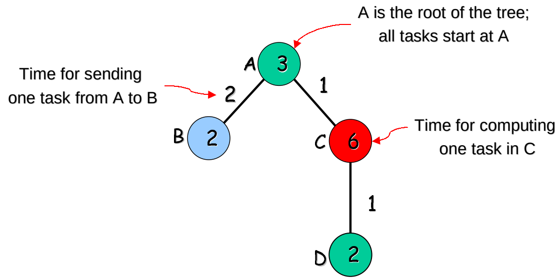
Makespan Maximize total number of tasks processed within a time-bound

Steady state Determine *periodic task allocation* which maximizes total throughput

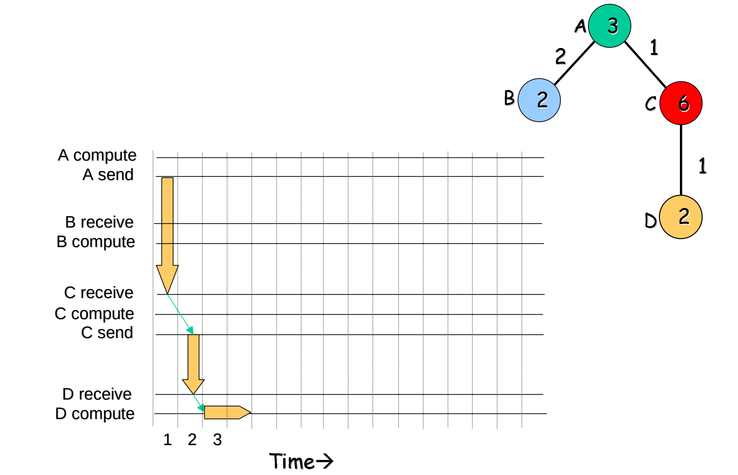
Example


Data starts here
My computer
Intermediate nodes can compute too
Internet Gateway
Cluster Host
Super-computer
Partner site
Participating PC's and workstations

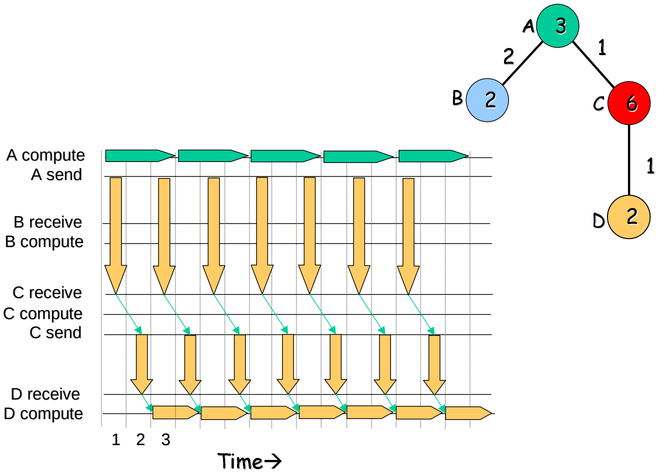
Example


A is the root of the tree;
all tasks start at A
Time for sending
one task from A to B
A
3
2
1
B
2
C
6
Time for computing
one task in C
1
D
2

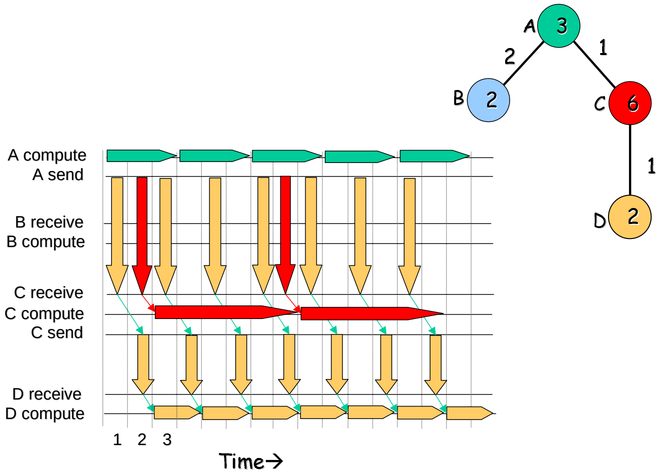
Example


A
3
2
1
B
2
C
6
1
D
2
A compute
A send
B receive
B compute
C receive
C compute
C send
D receive
D compute
1
2
3
Time→

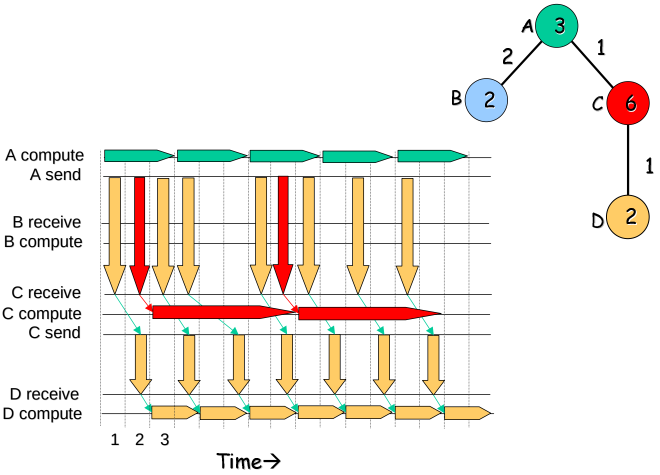
Example


A
3
2
1
B
2
C
6
1
D
2
A compute
A send
B receive
B compute
C receive
C compute
C send
D receive
D compute
1 2 3
Time→

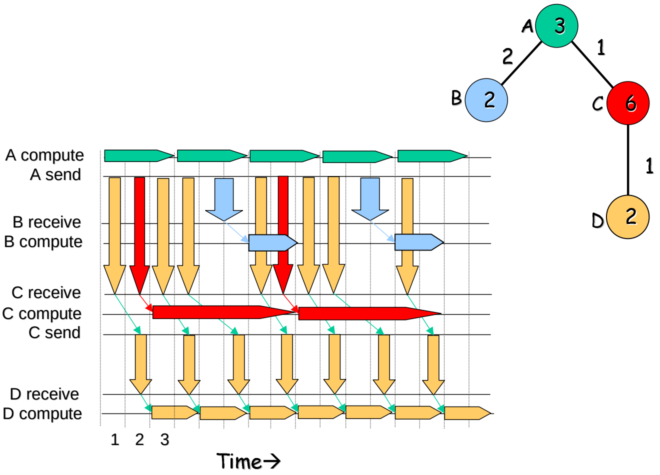
Example


A
3
2
1
B
2
C
6
1
D
2
A compute
A send
B receive
B compute
C receive
C compute
C send
D receive
D compute
1
2
3
Time→

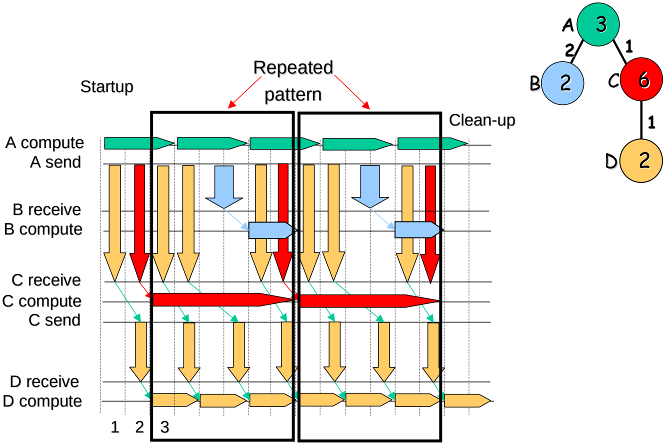
Example


A compute
A send
B receive
B compute
C receive
C compute
C send
D receive
D compute
1
2
3
Time→
A
3
2
1
B
2
C
6
1
D
2

Example

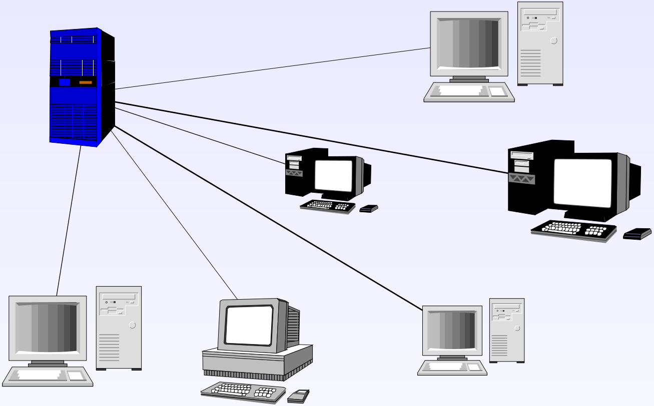

A compute
A send
B receive
B compute
C receive
C compute
C send
D receive
D compute
1
2
3
Time→
A
3
2
1
B
2
C
6
1
D
2

Example



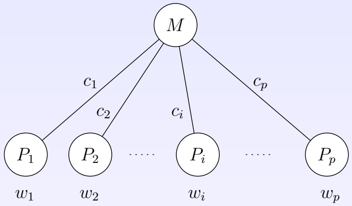
Steady-state: 7 tasks every 6 time units

Solution for star-shaped platforms



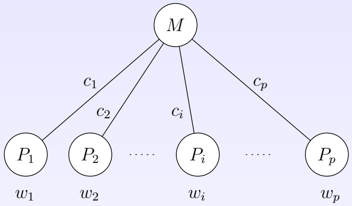
- Communication links between master and workers have *different* bandwidths
- Workers have *different* computing power

Rule of the game



- Master sends tasks to workers sequentially, and without preemption
- Full computation/communication overlap for each worker
- Worker P_i receives a task in c_i time-units
- Worker P_i processes a task in w_i time-units

Equations



- Worker P_i executes α_i tasks per time-unit
- Computations: $\alpha_i w_i \leq 1$
- Communications: $\sum_i \alpha_i c_i \leq 1$
- Objective: maximize throughput

$$\rho = \sum_i \alpha_i$$

Solution

- Faster-communicating workers first: $c_1 \leq c_2 \leq \dots$
- Make full use of first q workers, where q largest index s.t.

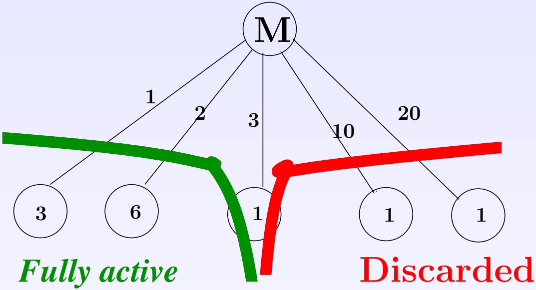
$$\sum_{i=1}^{q} \frac{c_i}{w_i} \leq 1$$

- Make partial use of next worker P_{q+1}
- Discard other workers

Bandwidth-centric strategy

- Delegate work to the fastest communicating workers
- It doesn't matter if these workers are computing slowly
- Of course, slow workers will not contribute much to the overall throughput

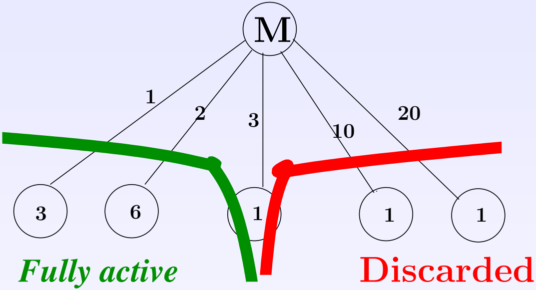
Example



Tasks	Communication	Computation
6 tasks to P_1	$6c_1 = 6$	$6w_1 = 18$
3 tasks to P_2	$3c_2 = 6$	$3w_2 = 18$
2 tasks to P_3	$2c_3 = 6$	$2w_3 = \mathbf{2}$

11 tasks every 18 time-units ($\rho = 11/18 \approx 0.6$)

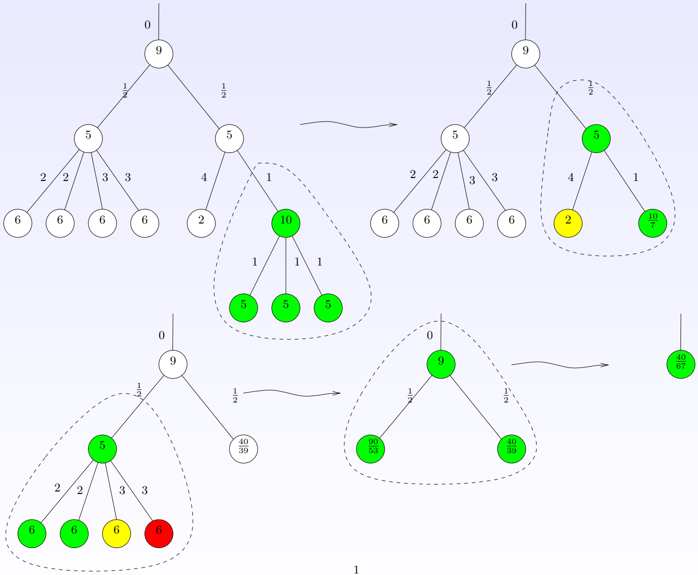
Example



☺ Compare to purely greedy (demand-driven) strategy!
5 tasks every 36 time-units ($\rho = 5/36 \approx 0.14$)

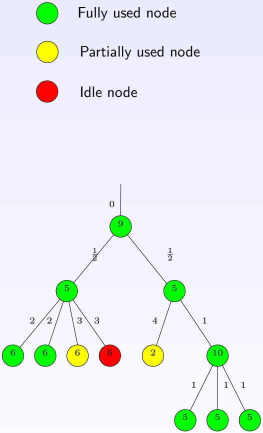
Even if resources are cheap and abundant,
resource selection is key to performance

Extension to trees



1

Extension to trees



Resource selection based on **local** information (children)

Does this really work?

- Can we deal with arbitrary platforms (including cycles)?
- Can we deal with return messages?
- In fact, can we deal with more complex applications (arbitrary collections of DAGs)?

Does this really work?

- Can we deal with arbitrary platforms (including cycles)? Yes
- Can we deal with return messages?
- In fact, can we deal with more complex applications (arbitrary collections of DAGs)?

Does this really work?

- Can we deal with arbitrary platforms (including cycles)? Yes
- Can we deal with return messages?
- In fact, can we deal with more complex applications (arbitrary collections of DAGs)?

Does this really work?

- Can we deal with arbitrary platforms (including cycles)? Yes
- Can we deal with return messages? Yes
- In fact, can we deal with more complex applications (arbitrary collections of DAGs)?

Does this really work?

- Can we deal with arbitrary platforms (including cycles)? Yes
- Can we deal with return messages? Yes
- In fact, can we deal with more complex applications (arbitrary collections of DAGs)?

Does this really work?

- Can we deal with arbitrary platforms (including cycles)? Yes
- Can we deal with return messages? Yes
- In fact, can we deal with more complex applications (arbitrary collections of DAGs)? Yes, I mean, almost!

LP formulation still works well ...

$$T_m \xrightarrow{\text{file } e_{mn}} T_n \qquad P_j \xrightarrow{c_{ji}} \underset{w_i}{P_i} \xrightarrow{c_{ik}} P_k$$

Conservation law

$$\forall m,n \quad \sum_j \text{sent}(P_j \to P_i, e_{mn}) + \text{executed}(P_i, T_m)$$
$$= \text{executed}(P_i, T_n) + \sum_k \text{sent}(P_i \to P_k, e_{mn})$$

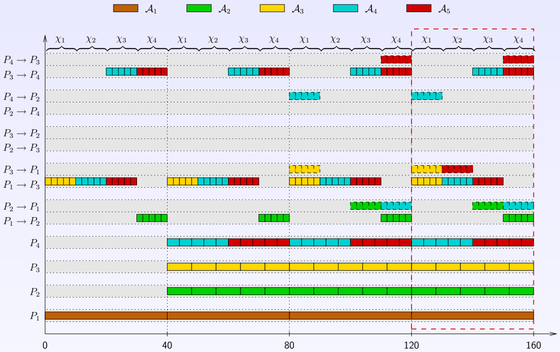
Computations

$$\sum_m \text{executed}(P_i, T_m) \times \text{flops}(T_m) \times w_i \leq 1$$

Outgoing communications

$$\sum_{m,n} \sum_j \text{sent}(P_j \to P_i, e_{mn}) \times \text{bytes}(e_{mn}) \times c_{ij} \leq 1$$

. . . but schedule reconstruction is harder



- ☺ Actual (cyclic) schedule obtained in polynomial time
- ☺ Asymptotic optimality
- ☹ A couple of practical problems (large period, # buffers)
- ☹ No **local** scheduling policy

The beauty of steady-state scheduling

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering/allocation of tasks/messages not needed
- Characterize resource activity during each time-unit:
  - which (rational) fraction of time is spent computing for which application?
  - which (rational) fraction of time is spent receiving or sending to which neighbor?

Efficiency Optimal throughput $\Rightarrow$ optimal schedule (up to a constant number of tasks)

Periodic schedule, described in compact form
$\Rightarrow$ compiling a loop instead of a DAG!

The beauty of steady-state scheduling

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering/allocation of tasks/messages not needed
- Characterize resource activity during each time-unit:
  - which (rational) fraction of time is spent computing for which application?
  - which (rational) fraction of time is spent receiving or sending to which neighbor?

Efficiency Optimal throughput $\Rightarrow$ optimal schedule (up to a constant number of tasks)

Periodic schedule, described in compact form
$\Rightarrow$ compiling a loop instead of a DAG!

Bibliography – Master-worker tasking

- Steady-state scheduling:
  Scheduling strategies for master-worker tasking on heterogeneous processor platforms, C. Banino et al., IEEE TPDS 15, 4 (2004), 319-330
- With bounded multi-port model:
  Distributed adaptive task allocation in heterogeneous computing environments to maximize throughput, B. Hong and V.K. Prasanna, IEEE IPDPS (2004), 52b
- With several applications:
  Centralized versus distributed schedulers for multiple bag-of-task applications, presented yesterday!

Outline

Broadcasting data

- Key collective communication operation
- Start: one processor has the data
- End: all processors own a copy
- Vast literature about broadcast, `MPI_Bcast`
- Standard approach: use a spanning tree
- Finding the best spanning tree: NP-Complete problem (even in the telephone model)

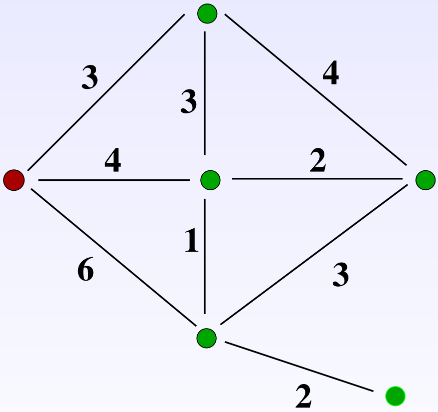
Broadcasting data

- Key collective communication operation
- Start: one processor has the data
- End: all processors own a copy
- Vast literature about broadcast, `MPI_Bcast`
- Standard approach: use a spanning tree
- Finding the best spanning tree: NP-Complete problem (even in the telephone model)

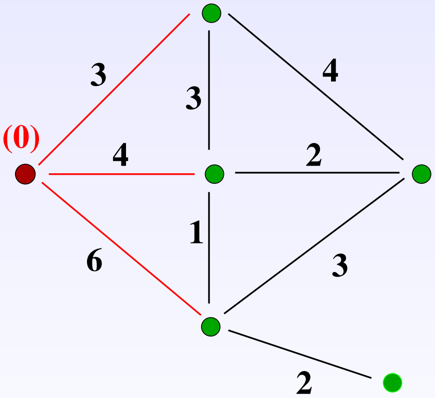
Broadcasting data

- Key collective communication operation
- Start: one processor has the data
- End: all processors own a copy
- Vast literature about broadcast, `MPI_Bcast`
- Standard approach: use a spanning tree
- Finding the best spanning tree: NP-Complete problem (even in the telephone model)

Heuristic: Earliest completing edge first (ECEF)

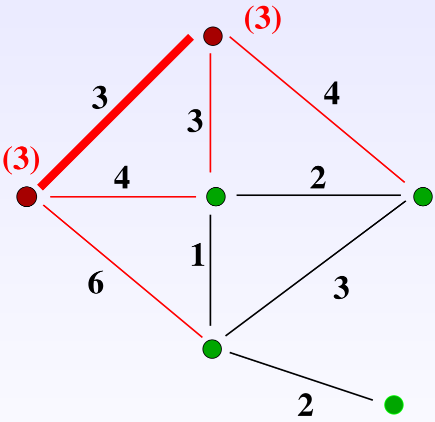


Heuristic: Earliest completing edge first (ECEF)



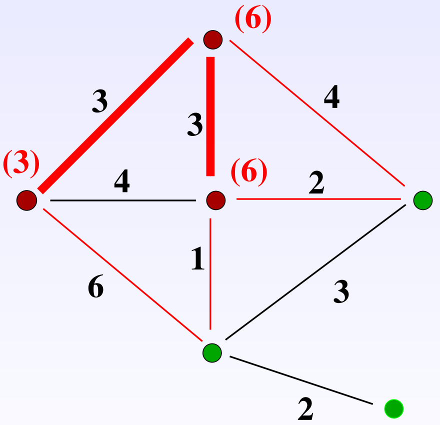
Next node: minimize $(R_i) + c_{ij}$, $P_j \notin \mathcal{T}$

Heuristic: Earliest completing edge first (ECEF)



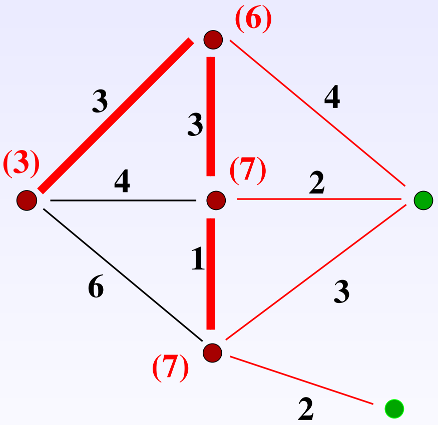
Next node: minimize $(R_i) + c_{ij}$, $P_j \notin \mathcal{T}$

Heuristic: Earliest completing edge first (ECEF)



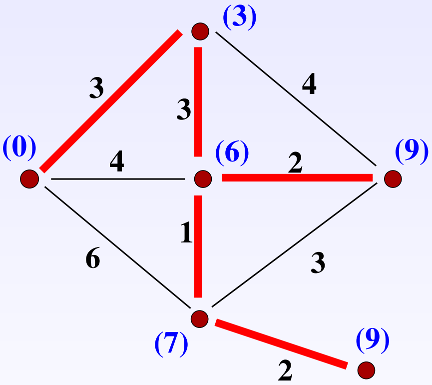
Next node: minimize $(R_i) + c_{ij}$, $P_j \notin \mathcal{T}$

Heuristic: Earliest completing edge first (ECEF)



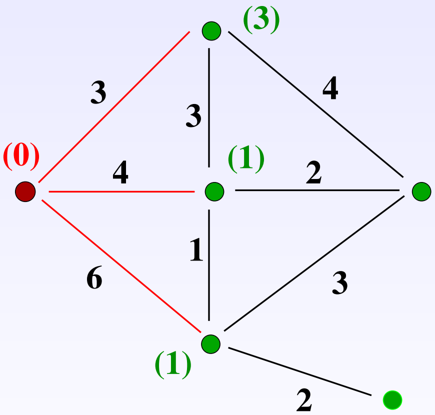
Next node: minimize $\color{red}{(R_i)} + c_{ij}$, $P_j \notin \mathcal{T}$

Heuristic: Earliest completing edge first (ECEF)



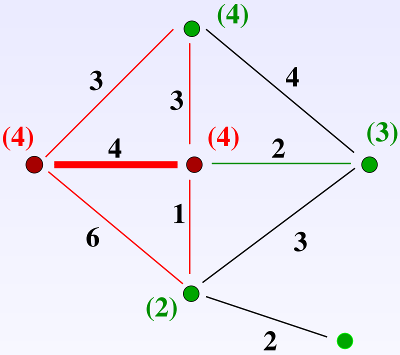
Broadcast finishing times (t)

Heuristic: Look-ahead (LA)



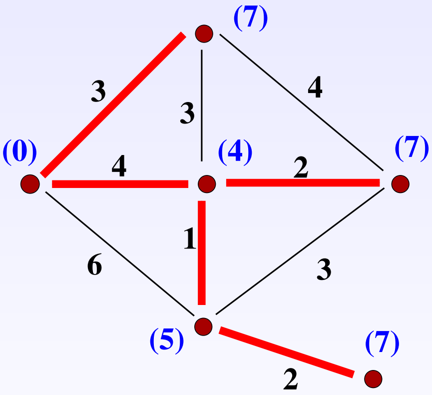
Next node: minimize $(R_i) + c_{ij} + (\min c_{jk})$, $P_j, P_k \notin \mathcal{T}$

Heuristic: Look-ahead (LA)



Next node: minimize $(R_i) + c_{ij} + (\min c_{jk})$, $P_j, P_k \notin \mathcal{T}$

Heuristic: Look-ahead (LA)



Broadcast finishing times (t)

Broadcasting longer messages

- Message size goes from L to, say, $10L$
- Communication costs scale from c_{ij} to $10c_{ij}$
- ECEF heuristic: broadcast time becomes 90
- LA heuristic: broadcast time becomes 70

Broadcasting longer messages

- Message size goes from L to, say, $10L$
- Communication costs scale from c_{ij} to $10c_{ij}$
- ECEF heuristic: broadcast time becomes 90
- LA heuristic: broadcast time becomes 70

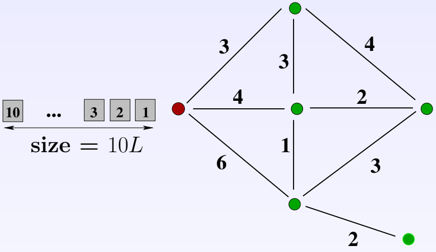
Broadcasting longer messages

- Message size goes from L … j, $10L$
- Communication costs scale from … to $10c_{ij}$
- ECEF heuristic: broadcast time becomes 90
- LA heuristic: broadcast time becomes 7…

Eh wait!

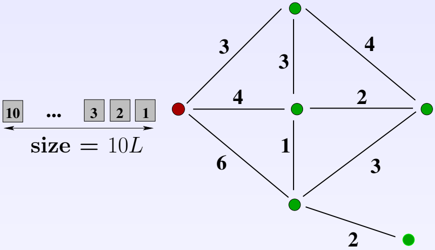
What about PIPELINING?!

Broadcasting longer messages



- Search spanning tree . . .
- Objective: minimize pipelined execution time

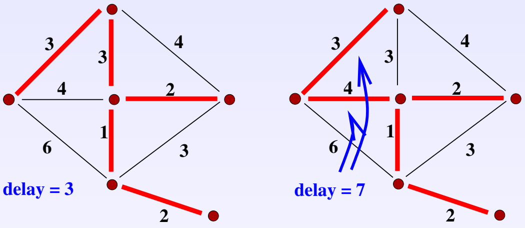
Broadcasting longer messages



- Delay = inverse of throughput
- Node delay = $\sum_{\text{children of node}}$ comm. times
- Tree delay = maximum node delay
- Pipelined execution time:

  (# edges in longest path + #packets) × tree delay
- Objective: minimize tree delay

Back to the example

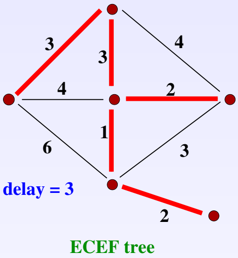


ECEF tree

LA tree

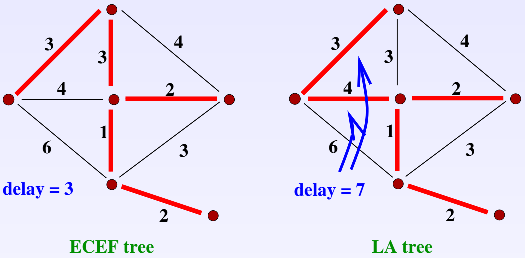
- ECEF tree turns out to have minimum delay (maximal throughput)
- Can we always find tree with optimal throughput?
- ☹ Problem is NP-complete
- ☺ Still, can design simple heuristics:
  SDIEF: smallest-delay-increase edge first

Back to the example



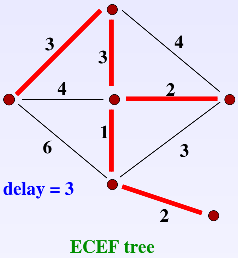
- ECEF tree turns out to have minimum delay (maximal throughput)
- Can we always find tree with optimal throughput?
- ☹ Problem is NP-complete
- ☺ Still, can design simple heuristics:
  SDIEF: smallest-delay-increase edge first

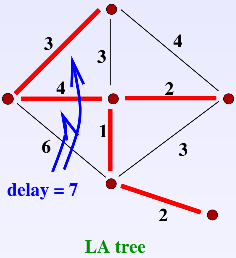
Back to the example



- ECEF tree turns out to have minimum delay (maximal throughput)
- Can we always find tree with optimal throughput?
- ☹ Problem is NP-complete
- ☺ Still, can design simple heuristics:
  SDIEF: smallest-delay-increase edge first

Back to the example





- ECEF tree turns out to have minimum delay (maximal throughput)
- Can we always find tree with optimal throughput?
- ☹ Problem is NP-complete
- ☺ Still, can design simple heuristics:
  SDIEF: smallest-delay-increase edge first

Assessing a broadcast strategy

- ☺ Finding optimal **set of** spanning tree**s** is polynomial: use LP formulation!
  ☹ Schedule reconstruction and packet management is harder with several trees
- Suggested approach:
  - Compute optimal throughput (several trees) with LP formulation
  - Run preferred heuristic to generate one or several "good" spanning trees
  - Stop refining when performance "reasonably" close to upper bound
- Will outperform MPI binomial spanning tree!

Assessing a broadcast strategy

- ☺ Finding optimal **set of** spanning tree**s** is polynomial: use LP formulation!
  ☹ Schedule reconstruction and packet management is harder with several trees
- Suggested approach:
  - Compute optimal throughput (several trees) with LP formulation
  - Run preferred heuristic to generate one or several "good" spanning trees
  - Stop refining when performance "reasonably" close to upper bound
- Will outperform MPI binomial spanning tree!

Bibliography – Broadcast

- Complexity:
  On broadcasting in heterogeneous networks, S. Khuller and Y.A. Kim, 15th ACM SODA (2004), 1011–1020
- Heuristics:
  Efficient collective communication in distributed heterogeneous systems, P.B. Bhat, C.S. Raghavendra and V.K. Prasanna, JPDC 63 (2003), 251–263
- Steady-state:
  Pipelining broadcasts on heterogeneous platforms, O. Beaumont et al., IEEE TPDS 16, 4 (2005), 300-313

Outline

Good news and bad news

- ☺ One-port model: first step towards designing realistic scheduling heuristics
- ☺ Steady-state circumvents complexity of scheduling problems . . . while deriving efficient (often asympotically optimal) scheduling algorithms
- ☹ Need to acquire a good knowledge of the platform graph
- ☹ Need to run extensive experiments or simulations

Good news and bad news

- ☺ One-port model: first step towards designing realistic scheduling heuristics
- ☺ Steady-state circumvents complexity of scheduling problems . . . while deriving efficient (often asympotically optimal) scheduling algorithms

- ☹ Need to acquire a good knowledge of the platform graph
- ☹ Need to run extensive experiments or simulations

Knowledge of the platform graph

- For regular problems, the *structure* of the task graph (nodes and edges) only depends upon the application, not upon the target platform
- Problems arise from *weights*, i.e. the estimation of execution and communication times
- Classical answer: *"use the past to predict the future"*
- Divide scheduling into phases, during which machine and network parameters are collected (with NWS)
  $\Rightarrow$ This information guides scheduling decisions for next phase
- Moving from heterogeneous clusters to computational grids causes further problems (even discovering the characteristics of the surrounding computing resources may prove a difficult task)

Knowledge of the platform graph

- For regular problems, the *structure* of the task graph (nodes and edges) only depends upon the application, not upon the target platform
- Problems arise from *weights*, i.e. the estimation of execution and communication times
- Classical answer: *"use the past to predict the future"*
- Divide scheduling into phases, during which machine and network parameters are collected (with NWS)
  $\Rightarrow$ This information guides scheduling decisions for next phase

- Moving from heterogeneous clusters to computational grids causes further problems (even discovering the characteristics of the surrounding computing resources may prove a difficult task)

Knowledge of the platform graph

- For regular problems, the *structure* of the task graph (nodes and edges) only depends upon the application, not upon the target platform
- Problems arise from *weights*, i.e. the estimation of execution and communication times
- Classical answer: *"use the past to predict the future"*
- Divide scheduling into phases, during which machine and network parameters are collected (with NWS)
  $\Rightarrow$ This information guides scheduling decisions for next phase
- Moving from heterogeneous clusters to computational grids causes further problems (even discovering the characteristics of the surrounding computing resources may prove a difficult task)

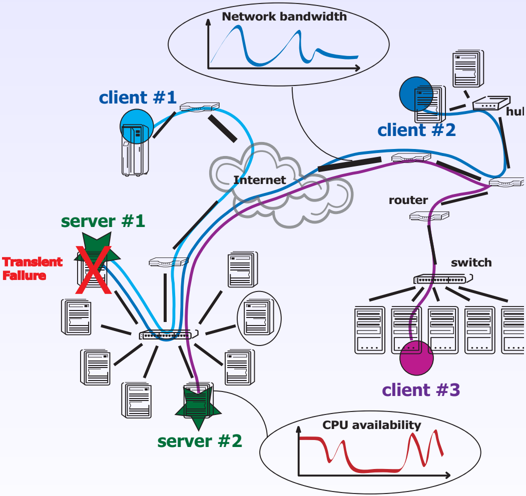
Experiments versus simulations

- Real experiments difficult to drive (genuine instability of non-dedicated platforms)
- Simulations ensure reproducibility of measured data
- Key issue: run simulations against a realistic environment
- *Trace-based simulation*: record platform parameters today, and simulate the algorithms tomorrow, against recorded data
- Use SimGrid, an event-driven simultation toolkit

Experiments versus simulations

- Real experiments difficult to drive (genuine instability of non-dedicated platforms)
- Simulations ensure reproducibility of measured data
- Key issue: run simulations against a realistic environment
- *Trace-based simulation*: record platform parameters today, and simulate the algorithms tomorrow, against recorded data
- Use SimGrid, an event-driven simultation toolkit

SimGrid traces

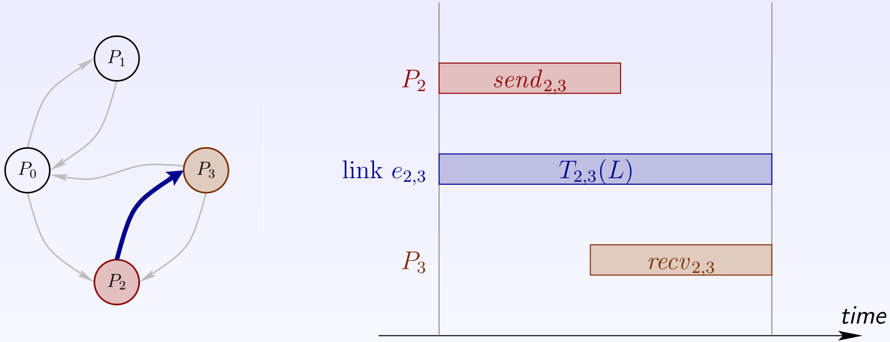


See http://simgrid.gforge.inria.fr/

Outline

Across physical links

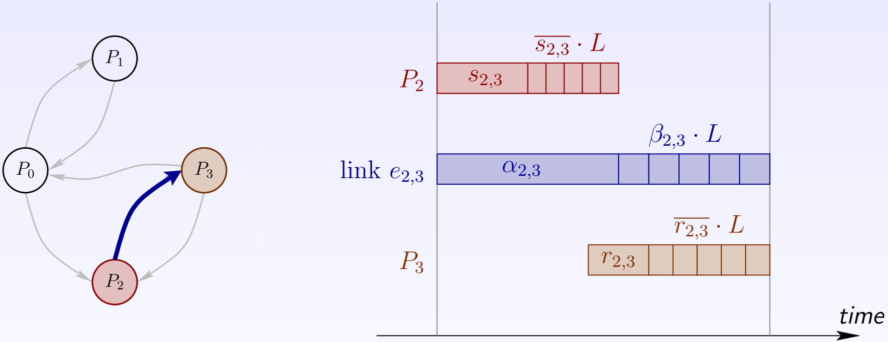
Network = directed graph $\mathcal{P} = (V, E)$



- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer

Across physical links

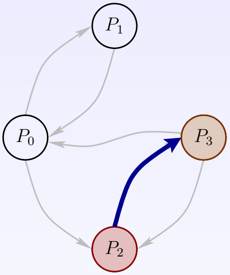
Network = directed graph $\mathcal{P} = (V, E)$

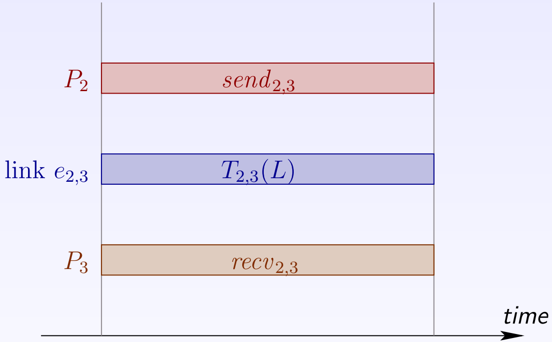


- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer

Across physical links

Network = directed graph $\mathcal{P} = (V, E)$

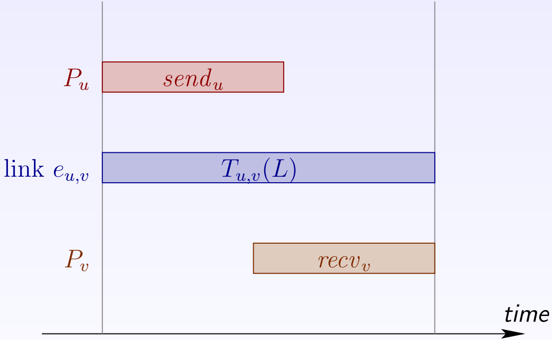




- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer

Multi-port

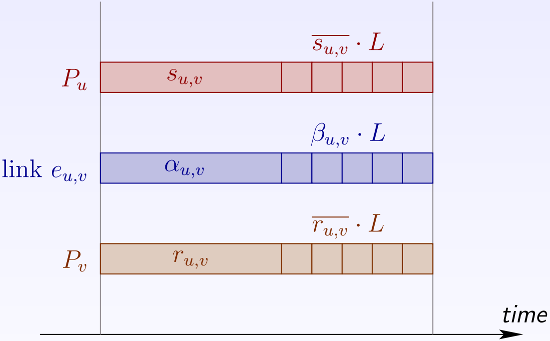
- Bar-Noy, Guha, Naor, Schieber:
  occupation time of sender P_u independent of target P_v



not *fully* multi-port model, but allows for starting a new transfer from P_u without waiting for previous one to finish

One-port

- Bhat, Raghavendra and Prasanna:
same parameters for sender P_u, link $e_{u,v}$ and receiver P_v



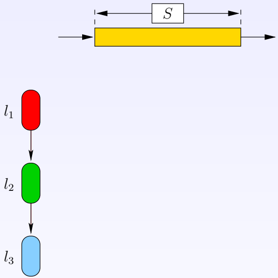
two flavors:
- bidirectional: simultaneous send *and* receive transfers allowed
- unidirectional: only one send or receive transfer at a given time-step

Store & Forward, WormHole, TCP

How to model a file transfer along a path?

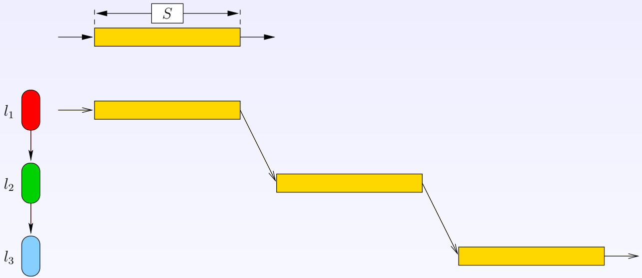
Store & Forward, WormHole, TCP

How to model a file transfer along a path?



Store & Forward, WormHole, TCP

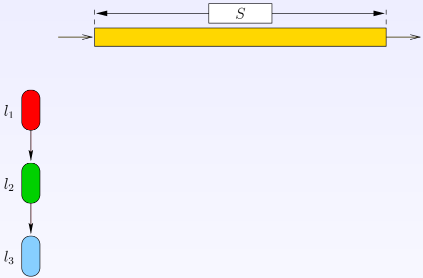
How to model a file transfer along a path?



Store & Forward : bad model for contention

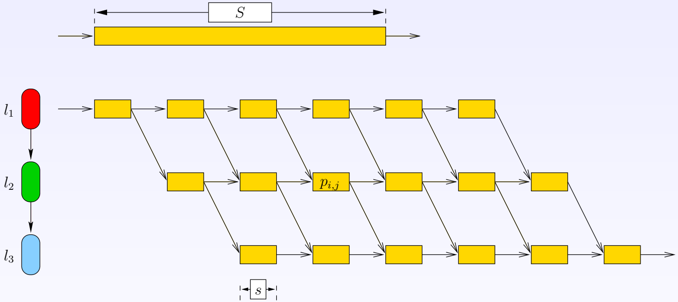
Store & Forward, WormHole, TCP

How to model a file transfer along a path?



Store & Forward, WormHole, TCP

How to model a file transfer along a path?



WormHole : computation intensive (packets), not that realistic

Store & Forward, WormHole, TCP

How to model a file transfer along a path?

$$\forall l \in \mathcal{L}, \quad \sum_{r \in \mathcal{R} \text{ s.t. } l \in r} \rho_r \leq c_l$$

Analytical model

Store & Forward, WormHole, TCP

How to model a file transfer along a path?

$$\forall l \in \mathcal{L}, \quad \sum_{r \in \mathcal{R} \text{ s.t. } l \in r} \rho_r \leq c_l$$

Max-Min Fairness maximize $\min_{r \in \mathcal{R}} \rho_r$

Proportional Fairness maximize $\sum_{r \in \mathcal{R}} \rho_r \log(\rho_r)$

MCT minimization maximize $\min_{r \in \mathcal{R}} \frac{1}{\rho_r}$

TCP behavior Close to max-min.

In SimGrid: max-min + bound by $1/RTT$

Bandwidth sharing

- Traditional assumption: Fair Sharing
- Open i TCP connections, receive $bw(i)$ bandwidth per connection
- $bw(i) = bw(1)/i$ on a LAN
- Experimental evidence $\rightarrow bw(i) = bw(1)$ on a WAN
- Backbone links have so many connections that interference among a few selected connections is negligible
- Better model: $bw(i) = \dfrac{bw(1)}{1 + (i-1).\gamma}$
- $\gamma = 1$ for a perfect LAN, $\gamma = 0$ for a perfect WAN

Bandwidth sharing

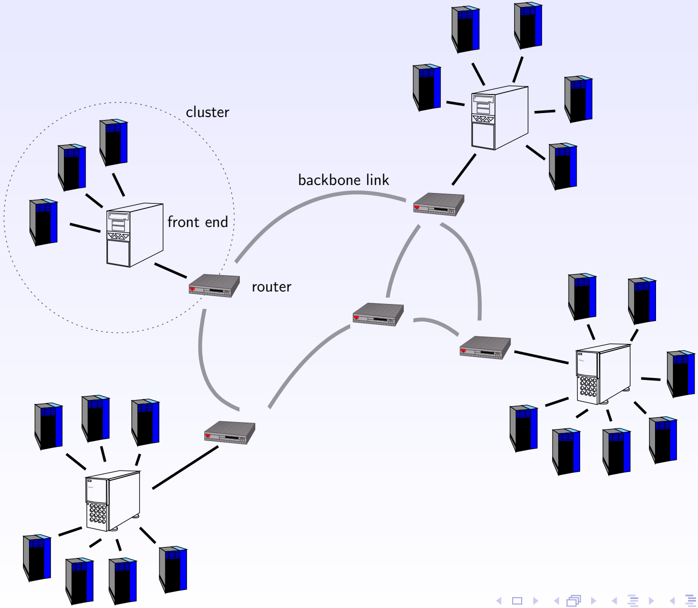
- Traditional assumption: Fair Sharing
- Open i TCP connections, receive $bw(i)$ bandwidth per connection
- $bw(i) = bw(1)/i$ on a LAN
- Experimental evidence $\rightarrow bw(i) = bw(1)$ on a WAN
- Backbone links have so many connections that interference among a few selected connections is negligible
- Better model: $bw(i) = \dfrac{bw(1)}{1 + (i-1).\gamma}$
- $\gamma = 1$ for a perfect LAN, $\gamma = 0$ for a perfect WAN

Bandwidth sharing

- Traditional assumption: Fair Sharing
- Open i TCP connections, receive $bw(i)$ bandwidth per connection
- $bw(i) = bw(1)/i$ on a LAN
- Experimental evidence $\rightarrow bw(i) = bw(1)$ on a WAN
- Backbone links have so many connections that interference among a few selected connections is negligible
- Better model: $bw(i) = \dfrac{bw(1)}{1 + (i-1).\gamma}$
- $\gamma = 1$ for a perfect LAN, $\gamma = 0$ for a perfect WAN

Outline

Sample large-scale platform



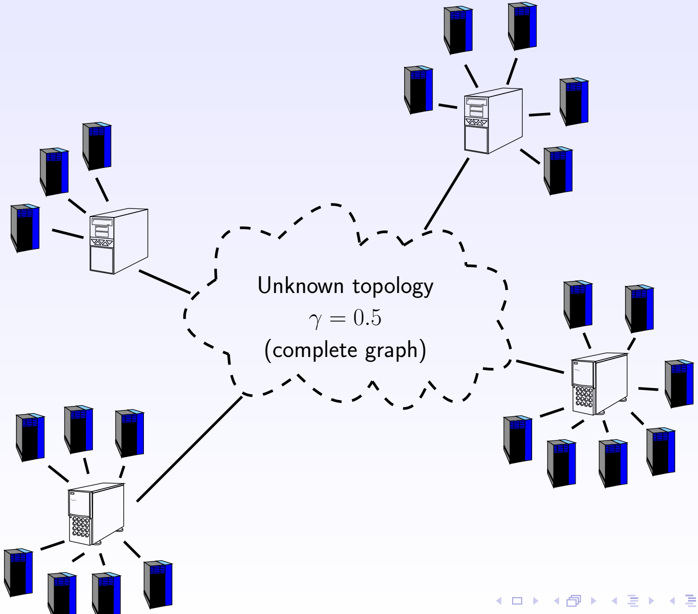
What topology?

- Generated (GT-ITM, BRITE, etc.) or obtained from monitoring?
  - Very complex (Layer 2 information)
  - Not clear that a scheduling algorithm could exploit/know all that information
- Need a simple model that is
  - More accurate than traditional models (e.g., LAN links, fully-connected)
  - Still amenable to analysis

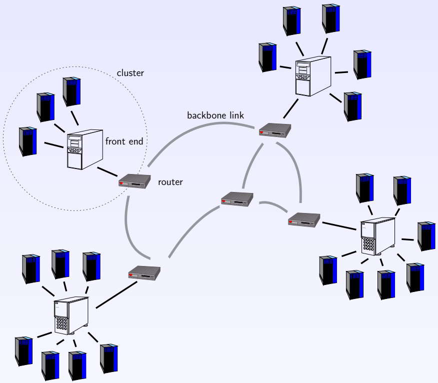
What topology?

- Generated (GT-ITM, BRITE, etc.) or obtained from monitoring?
  - Very complex (Layer 2 information)
  - Not clear that a scheduling algorithm could exploit/know all that information
- Need a simple model that is
  - More accurate than traditional models (e.g., LAN links, fully-connected)
  - Still amenable to analysis

What topology? (cont'd)



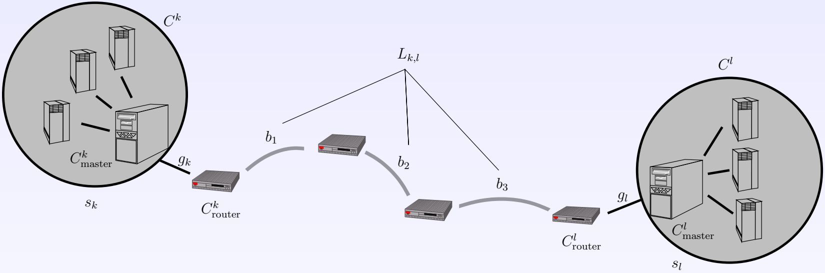
What topology? (cont'd)



Hierarchy + BW sharing, but assume knowledge of

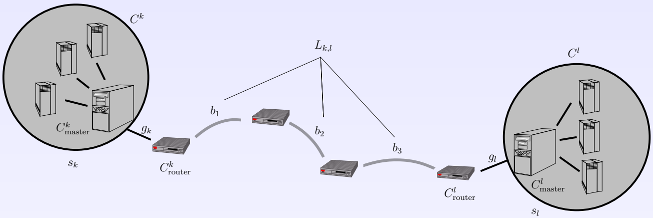
- Routing
- Backbone bandwidths
- CPU speeds

A first trial



Clusters and backbone links

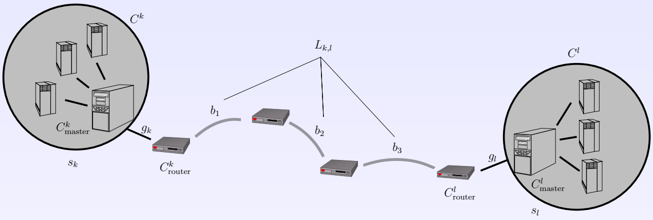
A first trial (cont'd)



Clusters

- K clusters C^k, $1 \leq k \leq K$
- C^k_{master} front-end processor
- C^k_{router} router to external world
- s_k cumulated speed of C^k
- g_k bandwidth of the LAN link $(\gamma = 1)$ from C^k_{master} to C^k_{router}

A first trial (cont'd)



Network

- Set $\mathcal{R}$ of routers and $\mathcal{B}$ of backbone links l_i
- bw(l_i) bandwidth available for a new connection
- max-connect(l_i) max. number of connections that can be opened
- Fixed routing: path $L_{k,l}$ of backbones from C^k_{router} to C^l_{router}

Bibliography

- NWS:
  The network weather service: a distributed resource performance forecasting service for metacomputing, R. Wolski, N.T. Spring and J. Hayes, Future Generation Computer Systems 15, 10 (1999), 757-768
- SimGrid:
  Scheduling distributed applications: the SimGrid simulation framework, A. Legrand, L. Marchal, and H. Casanova, 3rd IEEE CCGrid (2003), 138-145
- Bandwidth sharing:
  Bandwidth sharing: objectives and algorithms, L. Massoulié and J. Roberts, IEEE/ACM Trans. Networking 10, 3 (2002), 320-328

Outline

Scheduling multiple applications

- Large-scale platforms not likely to be exploited in dedicated mode/single application
- Investigate scenarios in which multiple applications are simultaneously executed on the platform
  $\Rightarrow$ competition for CPU and network resources

Target problem

1. Large complex platform: several clusters and backbone links
2. One (divisible load) application running on each cluster
3. Which fraction of the job to delegate to other clusters?
4. Applications have different communication-to-computation ratios
5. How to ensure fair scheduling and good resource utilization?

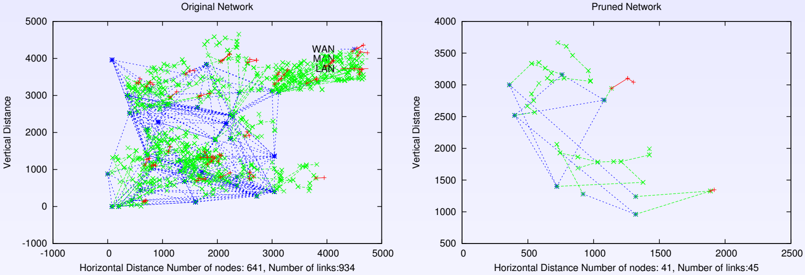
Linear program

$$
\begin{array}{l}
\textsc{Maximize} \;\; \min_k \left\{ \frac{\alpha_k}{\pi_k} \right\}, \\
\textsc{under the constraints} \\
\left\{
\begin{array}{lll}
\text{(1a)} & \forall C^k, & \sum_l \alpha_{k,l} = \alpha_k \\
\text{(1b)} & \forall C^k, & \sum_l \alpha_{l,k}.\tau_l \leq s_k \\
\text{(1c)} & \forall C^k, & \sum_{l \neq k} \alpha_{k,l}.\delta_k + \sum_{j \neq k} \alpha_{j,k}.\delta_j \leq g_k \\
\text{(1d)} & \forall l_i, & \sum_{l_i \in L_{k,l}} \beta_{k,l} \leq \text{max-connect}(l_i) \\
\text{(1e)} & \forall k,l, & \alpha_{k,l}.\delta_k \leq \beta_{k,l} \times g_{k,l} \\
\text{(1f)} & \forall k,l, & \alpha_{k,l} \geq 0 \\
\text{(1g)} & \forall k,l, & \beta_{k,l} \in \mathbb{N}
\end{array}
\right.
\end{array}
\qquad (1)
$$

Approach

- Solution to *rational* linear problem as comparator/upper bound
- Several heuristics, greedy and LP-based
- Use Tiers as topology generator, and then SimGrid

Methodology (cont'd)



	distribution
K	$5, 7, \ldots, 90$
$\log(bw(l_k))$, $\log(g_k)$	normal ($mean = \log(2000)$, $std = \log(10)$)
s_k	uniform, 1000 — 10000
max-connect, δ_k, τ_k, π_k	uniform, 1 — 10

Platform parameters used in simulation

Hints for implementation

- Participants sharing resources in a Virtual Organization
- Centralized broker managing applications and resources
- Broker gathers all parameters of LP program
- Priority factors
- Various policies and refinements possible
  $\Rightarrow$ e.g. fixed number of connections per application

Bibliography

- Tiers:
  Modeling Internet topology, K. Calvert, M. Doar and E.W. Zegura, IEEE Comm. Magazine 35, 6 (1997), 160-163
- Scheduling multiple applications:
  A realistic network/application model for scheduling divisible loads on large-scale platforms, L. Marchal et al., 19th IEEE IPDPS (2005)

Outline

Key advantages of steady-state scheduling

Simplicity

- From local equations to global behavior
- Throughput characterized from activity variables

Efficiency

- Periodic schedule, described in compact form
- Asymptotic optimality

Adaptability

- Record observed performance during current period
- Inject information to compute schedule for next period
- React on the fly to resource availability variations

Open problems

- Decentralized scheduling
  - From local strategies to provably good performance?
  - Adapt Awerbuch-Leighton algorithm for multicommodity flows?
- Concurrent scheduling
  - Multi-criteria and fairness?
  - Adapt economic models and buzz-words (e.g., Nash equilibrium)?

Scheduling for heterogeneous platforms

- If the platform is well identified and relatively stable, try to:
  (i) accurately model the (expected) hierarchical structure of the platform
  (ii) design scheduling algorithms well-suited to this hierarchical structure

- If the platform is not stable enough, or if it evolves too fast, dynamic schedulers are the only option

- Otherwise, grab the opportunity to **inject some static knowledge into dynamic schedulers**:
  ☹ Is this opportunity a niche?
  ☺ Does it encompass a wide range of applications?

Answer to first comment

Comment

Scheduling is "this thing that people in academia like to think about but that people who do real stuff sort of ignore"

Answer

☹ Thank you for your attention.

Other comments or questions?

Answer to first comment

Comment

Scheduling is "this thing that people in academia like to think about but that people who do real stuff sort of ignore"

Answer

☹ Thank you for your attention.

Other comments or questions?