Centralized versus distributed schedulers for multiple bag-of-task applications

O. Beaumont, L. Carter, J. Ferrante,
A. Legrand, L. Marchal and Y. Robert

Laboratoire LaBRI, CNRS Bordeaux, France

Dept. of Computer Science and Engineering,
University of California, San Diego, USA

Laboratoire ID-IMAG, CNRS-INRIA Grenoble, France

Laboratoire de l'Informatique du Parallélisme
École Normale Supérieure de Lyon, France

November 2005

Outline

Outline

Motivation

- Multiple bag-of-tasks competing for CPU and network resources
- Steady-state scheduling
- Assessing centralized versus decentralized approaches

Motivation

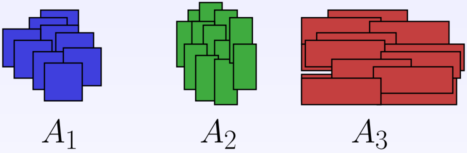
- Multiple bag-of-tasks competing for CPU and network resources
- Steady-state scheduling
- Assessing centralized versus decentralized approaches

Motivation

- Multiple bag-of-tasks competing for CPU and network resources
- Steady-state scheduling
- Assessing centralized versus decentralized approaches

Introduction – Applications

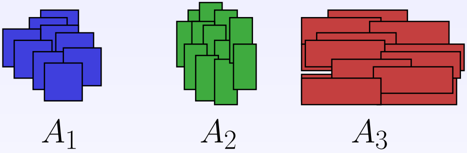
- Multiple applications:
  - each consisting in a large number of same-size independent tasks
  - all competing for CPU and network resources



- Different communication and computation demands for different applications
- Important parameter: $\frac{\text{communication size}}{\text{computation size}}$

Introduction – Applications

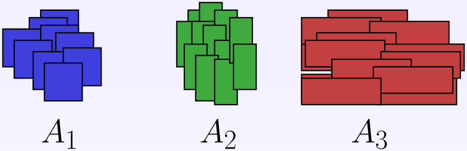
- Multiple applications:
  - each consisting in a large number of same-size independent tasks
  - all competing for CPU and network resources



- Different communication and computation demands for different applications
- Important parameter: $\frac{\text{communication size}}{\text{computation size}}$

Introduction – Applications

- Multiple applications:
  - each consisting in a large number of same-size independent tasks
  - all competing for CPU and network resources

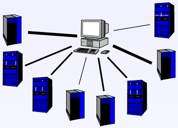


- Different communication and computation demands for different applications
- Important parameter: $\frac{\text{communication size}}{\text{computation size}}$

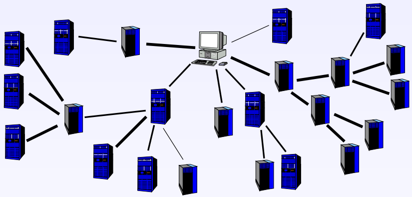
Introduction – Platform

- Target platform: master-worker

star network



tree network



- Master holds all tasks initially

Introduction – Goals

- Maximize throughput
- Maintain balanced execution between applications (*fairness*)
- Scheduling decisions:
  - at master: which applications to assign to which subtree
  - at nodes (tree): which tasks to forward to which children
- Objective function:
  - priority weight: $w^{(k)}$ for application A_k
  - throughput:
    $\alpha^{(k)}$ = number of tasks of type k computed per time-unit
  - MAX-MIN fairness: $\textsc{Maximize} \min_k \left\{ \frac{\alpha^{(k)}}{w^{(k)}} \right\}$.

Introduction – Goals

- Maximize throughput
- Maintain balanced execution between applications (*fairness*)
- Scheduling decisions:
  - at master: which applications to assign to which subtree
  - at nodes (tree): which tasks to forward to which children
- Objective function:
  - priority weight: $w^{(k)}$ for application A_k
  - throughput: $\alpha^{(k)}$ = number of tasks of type k computed per time-unit
  - MAX-MIN fairness: $\textsc{Maximize} \min_k \left\{ \frac{\alpha^{(k)}}{w^{(k)}} \right\}$.

Introduction – Goals

- Maximize throughput
- Maintain balanced execution between applications (*fairness*)
- Scheduling decisions:
  - at master: which applications to assign to which subtree
  - at nodes (tree): which tasks to forward to which children
- Objective function:
  - priority weight: $w^{(k)}$ for application A_k
  - throughput:
    $\alpha^{(k)} =$ number of tasks of type k computed per time-unit
  - MAX-MIN fairness: $\text{MAXIMIZE} \min_k \left\{ \frac{\alpha^{(k)}}{w^{(k)}} \right\}$.

Introduction – Goals

- Maximize throughput
- Maintain balanced execution between applications (*fairness*)
- Scheduling decisions:
  - at master: which applications to assign to which subtree
  - at nodes (tree): which tasks to forward to which children
- Objective function:
  - priority weight: $w^{(k)}$ for application A_k
  - throughput:
    $\alpha^{(k)} =$ number of tasks of type k computed per time-unit
  - MAX-MIN fairness: $\textsc{Maximize} \min_k \left\{ \frac{\alpha^{(k)}}{w^{(k)}} \right\}$.

Introduction – Strategies

- Centralized strategies
  - central scheduler at master
  - complete and reliable knowledge of the platform
  - optimal schedule (Linear Programming formulation)
  - reasonable for small platforms
- Decentralized strategies
  - more realistic for large scale platforms
  - only local information available at each node (neighbors)
  - assume limited memory at each node
  - decentralized heuristics

Introduction – Strategies

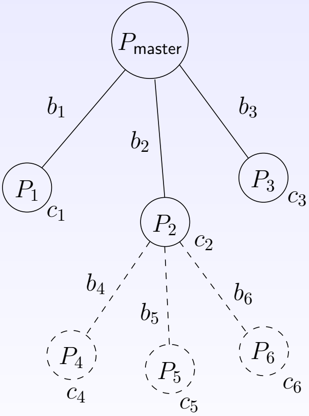
- Centralized strategies
  - central scheduler at master
  - complete and reliable knowledge of the platform
  - optimal schedule (Linear Programming formulation)
  - reasonable for small platforms
- Decentralized strategies
  - more realistic for large scale platforms
  - only local information available at each node (neighbors)
  - assume limited memory at each node
  - decentralized heuristics

Related Work

- See technical report RR-2005-45 at graal.ens-lyon.fr/~yrobert
- Topics:
  - Platform models
  - Steady-state scheduling
  - Scheduling (multiple) divisible loads
  - Master-slave on the computational grid
  - Fairness

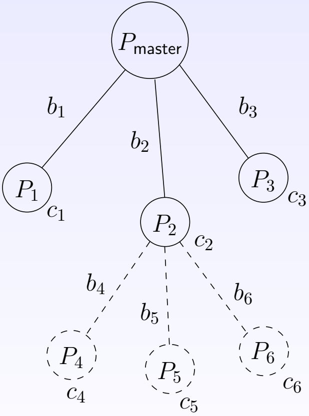
Outline

Platform Model



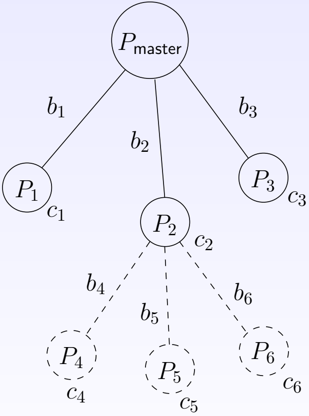
- Star or tree network
- Workers $P_1, \ldots, P_p$, master P_{master}
- Parent of P_u: $P_{p(u)}$
- Bandwidth of link $P_u \rightarrow P_{p(u)}$: b_u
- Computing speed of P_u: c_u
- Full communication/computation overlap
- One-port model

Platform Model



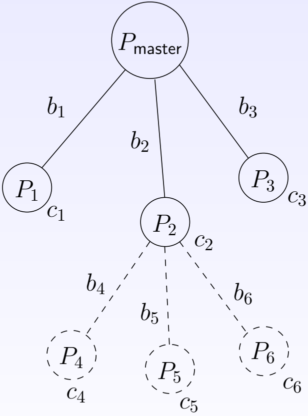
- Star or tree network
- Workers $P_1, \ldots, P_p$, master P_{master}
- Parent of P_u: $P_{p(u)}$
- Bandwidth of link $P_u \rightarrow P_{p(u)}$: b_u
- Computing speed of P_u: c_u
- Full communication/computation overlap
- One-port model

Platform Model



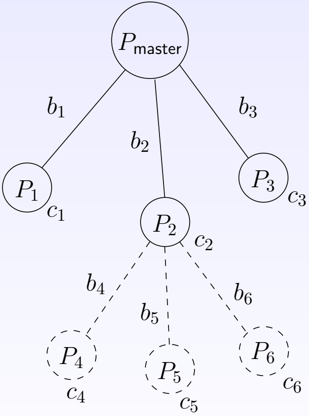
- Star or tree network
- Workers $P_1, \ldots, P_p$, master P_{master}
- Parent of P_u: $P_{p(u)}$
- Bandwidth of link $P_u \rightarrow P_{p(u)}$: b_u
- Computing speed of P_u: c_u
- Full communication/computation overlap
- One-port model

Platform Model



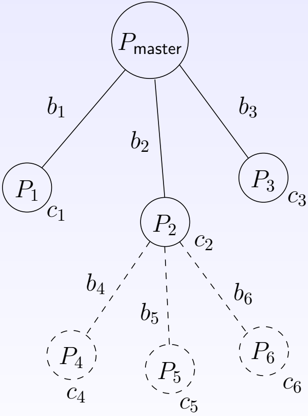
- Star or tree network
- Workers $P_1, \ldots, P_p$, master P_{master}
- Parent of P_u: $P_{p(u)}$
- Bandwidth of link $P_u \rightarrow P_{p(u)}$: b_u
- Computing speed of P_u: c_u
- Full communication/computation overlap
- One-port model

Platform Model



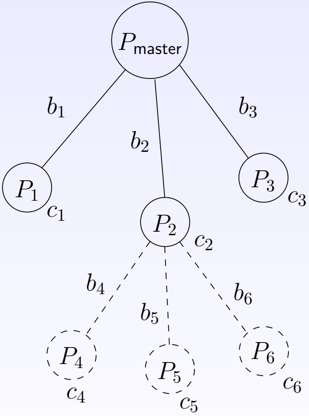
- Star or tree network
- Workers $P_1, \ldots, P_p$, master P_{master}
- Parent of P_u: $P_{p(u)}$
- Bandwidth of link $P_u \rightarrow P_{p(u)}$: b_u
- Computing speed of P_u: c_u
- Full communication/computation overlap
- One-port model

Platform Model



- Star or tree network
- Workers $P_1, \ldots, P_p$, master P_{master}
- Parent of P_u: $P_{p(u)}$
- Bandwidth of link $P_u \rightarrow P_{p(u)}$: b_u
- Computing speed of P_u: c_u
- Full communication/computation overlap
- One-port model

Platform Model



- Star or tree network
- Workers $P_1, \ldots, P_p$, master P_{master}
- Parent of P_u: $P_{p(u)}$
- Bandwidth of link $P_u \rightarrow P_{p(u)}$: b_u
- Computing speed of P_u: c_u
- Full communication/computation overlap
- One-port model

Application Model

- K applications $A_1, \ldots, A_k$
- Priority weights $w^{(k)}$: $w^{(1)} = 3$ and $w^{(2)} = 1 \iff$ process 3 tasks of type 1 per task of type 2
- For each task of A_k:
  - processing cost $c^{(k)}$ (MFlops)
  - communication cost $b^{(k)}$ (MBytes)
- Communication for input data only (no result message)
- *communication-to-computation* ratio (CCR): $\frac{b^{(k)}}{c^{(k)}}$

Application Model

- K applications $A_1, \ldots, A_k$
- Priority weights $w^{(k)}$: $w^{(1)} = 3$ and $w^{(2)} = 1 \iff$ process 3 tasks of type 1 per task of type 2
- For each task of A_k:
  - processing cost $c^{(k)}$ (MFlops)
  - communication cost $b^{(k)}$ (MBytes)
- Communication for input data only (no result message)
- *communication-to-computation* ratio (CCR): $\frac{b^{(k)}}{c^{(k)}}$

Application Model

- K applications $A_1, \ldots, A_k$
- Priority weights $w^{(k)}$: $w^{(1)} = 3$ and $w^{(2)} = 1 \iff$ process 3 tasks of type 1 per task of type 2
- For each task of A_k:
  - processing cost $c^{(k)}$ (MFlops)
  - communication cost $b^{(k)}$ (MBytes)
- Communication for input data only (no result message)
- *communication-to-computation* ratio (CCR): $\frac{b^{(k)}}{c^{(k)}}$

Application Model

- K applications $A_1, \ldots, A_k$
- Priority weights $w^{(k)}$: $w^{(1)} = 3$ and $w^{(2)} = 1 \iff$ process 3 tasks of type 1 per task of type 2
- For each task of A_k:
  - processing cost $c^{(k)}$ (MFlops)
  - communication cost $b^{(k)}$ (MBytes)
- Communication for input data only (no result message)
- *communication-to-computation* ratio (CCR): $\frac{b^{(k)}}{c^{(k)}}$

Application Model

- K applications $A_1, \ldots, A_k$
- Priority weights $w^{(k)}$: $w^{(1)} = 3$ and $w^{(2)} = 1 \iff$ process 3 tasks of type 1 per task of type 2
- For each task of A_k:
  - processing cost $c^{(k)}$ (MFlops)
  - communication cost $b^{(k)}$ (MBytes)
- Communication for input data only (no result message)
- *communication-to-computation* ratio (CCR): $\frac{b^{(k)}}{c^{(k)}}$

Outline

Steady-state Scheduling

Background Approach pioneered by Bertsimas and Gamarnik

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering of tasks/messages not needed

  computing for which application?

  - which (rational) fraction of time is spent receiving or sending to which neighbor?

Efficiency Periodic schedule, described in compact form

Adaptability Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period

$\Rightarrow$ react on the fly to resource availability variations

Steady-state Scheduling

Background Approach pioneered by Bertsimas and Gamarnik

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering of tasks/messages not needed

computing for which application?
- which (rational) fraction of time is spent receiving or sending to which neighbor?

Efficiency Periodic schedule, described in compact form

Adaptability Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period
⇒ react on the fly to resource availability variations

Steady-state Scheduling

Background Approach pioneered by Bertsimas and Gamarnik

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering 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 Periodic schedule, described in compact form

Adaptability Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period
⇒ react on the fly to resource availability variations

Steady-state Scheduling

Background Approach pioneered by Bertsimas and Gamarnik

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering 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 Periodic schedule, described in compact form

Adaptability Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period
⇒ react on the fly to resource availability variations

Steady-state Scheduling

Background Approach pioneered by Bertsimas and Gamarnik

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering 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 Periodic schedule, described in compact form

Adaptability Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period
$\Rightarrow$ react on the fly to resource availability variations

Steady-state Scheduling

Background Approach pioneered by Bertsimas and Gamarnik

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering 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 Periodic schedule, described in compact form

Adaptability Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period
$\Rightarrow$ react on the fly to resource availability variations

Steady-state Scheduling

Background Approach pioneered by Bertsimas and Gamarnik

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering 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 Periodic schedule, described in compact form

Adaptability Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period

$\Rightarrow$ react on the fly to resource availability variations

Steady-state Scheduling

Background Approach pioneered by Bertsimas and Gamarnik

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering 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 Periodic schedule, described in compact form

Adaptability Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period

$\Rightarrow$ react on the fly to resource availability variations

Linear Program for a Star Network

- $\alpha_u^{(k)}$ = rational number of tasks of A_k executed by P_u every time-unit
- $\alpha_u^{(k)} = 0$ for all $A_k \iff P_u$ does not participate
- Constraint for computations by P_u:

$$\sum_k \alpha_u^{(k)} \cdot c^{(k)} \leqslant c_u$$

- Number of bytes sent to worker P_u: $\sum_{k=1}^{K} \alpha_u^{(k)} \cdot b^{(k)}$
- Constraint for communications from the master:

$$\sum_{u=1}^{p} \frac{\sum_{k=1}^{K} \alpha_u^{(k)} \cdot b^{(k)}}{b_u} \leqslant 1$$

- Throughput for application A_k: $\alpha^{(k)} = \sum_{u=1}^{p} \alpha_u^{(k)}$
- Objective:

$$\text{MAXIMIZE} \quad \min_k \frac{\alpha^{(k)}}{w^{(k)}}$$

Linear Program for a Star Network

- $\alpha_u^{(k)}$ = rational number of tasks of A_k executed by P_u every time-unit
- $\alpha_u^{(k)} = 0$ for all $A_k \iff P_u$ does not participate
- Constraint for computations by P_u:

$$\sum_k \alpha_u^{(k)} \cdot c^{(k)} \leqslant c_u$$

- Number of bytes sent to worker P_u: $\sum_{k=1}^{K} \alpha_u^{(k)} \cdot b^{(k)}$
- Constraint for communications from the master:

$$\sum_{u=1}^{p} \frac{\sum_{k=1}^{K} \alpha_u^{(k)} \cdot b^{(k)}}{b_u} \leqslant 1$$

- Throughput for application A_k: $\alpha^{(k)} = \sum_{u=1}^{p} \alpha_u^{(k)}$
- Objective:

$$\text{MAXIMIZE} \quad \min_k \frac{\alpha^{(k)}}{w^{(k)}}$$

Linear Program for a Star Network

- $\alpha_u^{(k)}$ = rational number of tasks of A_k executed by P_u every time-unit
- $\alpha_u^{(k)} = 0$ for all $A_k \iff P_u$ does not participate
- Constraint for computations by P_u:

$$\sum_k \alpha_u^{(k)} \cdot c^{(k)} \leqslant c_u$$

- Number of bytes sent to worker P_u: $\sum_{k=1}^{K} \alpha_u^{(k)} \cdot b^{(k)}$
- Constraint for communications from the master:

$$\sum_{u=1}^{p} \frac{\displaystyle\sum_{k=1}^{K} \alpha_u^{(k)} \cdot b^{(k)}}{b_u} \leqslant 1$$

- Throughput for application A_k: $\alpha^{(k)} = \sum_{u=1}^{p} \alpha_u^{(k)}$
- Objective:

$$\text{MAXIMIZE} \quad \min_k \frac{\alpha^{(k)}}{w^{(k)}}$$

Linear Program for a Star Network

- $\alpha_u^{(k)}$ = rational number of tasks of A_k executed by P_u every time-unit
- $\alpha_u^{(k)} = 0$ for all $A_k \iff P_u$ does not participate
- Constraint for computations by P_u:

$$\sum_k \alpha_u^{(k)} \cdot c^{(k)} \leqslant c_u$$

- Number of bytes sent to worker P_u: $\sum_{k=1}^{K} \alpha_u^{(k)} \cdot b^{(k)}$
- Constraint for communications from the master:

$$\sum_{u=1}^{p} \frac{\sum_{k=1}^{K} \alpha_u^{(k)} \cdot b^{(k)}}{b_u} \leqslant 1$$

- Throughput for application A_k: $\alpha^{(k)} = \sum_{u=1}^{p} \alpha_u^{(k)}$
- Objective:

$$\text{MAXIMIZE} \ \min_k \frac{\alpha^{(k)}}{w^{(k)}}$$

Linear Program for a Star Network

- $\alpha_u^{(k)}$ = rational number of tasks of A_k executed by P_u every time-unit
- $\alpha_u^{(k)} = 0$ for all $A_k \iff P_u$ does not participate
- Constraint for computations by P_u:

$$\sum_k \alpha_u^{(k)} \cdot c^{(k)} \leqslant c_u$$

- Number of bytes sent to worker P_u: $\sum_{k=1}^{K} \alpha_u^{(k)} \cdot b^{(k)}$
- Constraint for communications from the master:

$$\sum_{u=1}^{p} \frac{\displaystyle\sum_{k=1}^{K} \alpha_u^{(k)} \cdot b^{(k)}}{b_u} \leqslant 1$$

- Throughput for application A_k: $\alpha^{(k)} = \sum_{u=1}^{p} \alpha_u^{(k)}$
- Objective:

$$\text{MAXIMIZE} \quad \min_k \frac{\alpha^{(k)}}{w^{(k)}}$$

Linear Program for a Star Network

- $\alpha_u^{(k)}$ = rational number of tasks of A_k executed by P_u every time-unit
- $\alpha_u^{(k)} = 0$ for all $A_k \iff P_u$ does not participate
- Constraint for computations by P_u:

$$\sum_k \alpha_u^{(k)} \cdot c^{(k)} \leqslant c_u$$

- Number of bytes sent to worker P_u: $\sum_{k=1}^{K} \alpha_u^{(k)} \cdot b^{(k)}$
- Constraint for communications from the master:

$$\sum_{u=1}^{p} \frac{\displaystyle\sum_{k=1}^{K} \alpha_u^{(k)} \cdot b^{(k)}}{b_u} \leqslant 1$$

- Throughput for application A_k: $\alpha^{(k)} = \sum_{u=1}^{p} \alpha_u^{(k)}$
- Objective:

$$\text{MAXIMIZE} \quad \min_k \frac{\alpha^{(k)}}{w^{(k)}}$$

Linear Program for a Star Network

- $\alpha_u^{(k)}$ = rational number of tasks of A_k executed by P_u every time-unit
- $\alpha_u^{(k)} = 0$ for all $A_k \iff P_u$ does not participate
- Constraint for computations by P_u:

$$\sum_k \alpha_u^{(k)} \cdot c^{(k)} \leqslant c_u$$

- Number of bytes sent to worker P_u: $\sum_{k=1}^{K} \alpha_u^{(k)} \cdot b^{(k)}$
- Constraint for communications from the master:

$$\sum_{u=1}^{p} \frac{\displaystyle\sum_{k=1}^{K} \alpha_u^{(k)} \cdot b^{(k)}}{b_u} \leqslant 1$$

- Throughput for application A_k: $\alpha^{(k)} = \sum_{u=1}^{p} \alpha_u^{(k)}$
- Objective:

$$\text{MAXIMIZE} \quad \min_k \frac{\alpha^{(k)}}{w^{(k)}}$$

Reconstructing an Optimal Schedule

- Solution of linear program: $\alpha_u^{(k)} = \frac{p_{u,k}}{q_{u,k}}$, throughput ρ
- Set period length: $T_p = \text{lcm}\{q_{u,k}\}$
- During each period, send $n_u^{(k)} = \alpha_u^{(k)} \cdot T_{\text{period}}$ to each worker P_u
  $\Rightarrow$ periodic schedule with throughput ρ
- Initialization and clean-up phases
- Asymptotically optimal schedule (computes optimal number of tasks in time T, up to a constant independent of T)

Reconstructing an Optimal Schedule

- Solution of linear program: $\alpha_u^{(k)} = \frac{p_{u,k}}{q_{u,k}}$, throughput ρ
- Set period length: $T_p = \text{lcm}\{q_{u,k}\}$
- During each period, send $n_u^{(k)} = \alpha_u^{(k)} \cdot T_{\text{period}}$ to each worker P_u $\Rightarrow$ periodic schedule with throughput ρ
- Initialization and clean-up phases
- Asymptotically optimal schedule (computes optimal number of tasks in time T, up to a constant independent of T)

Reconstructing an Optimal Schedule

- Solution of linear program: $\alpha_u^{(k)} = \frac{p_{u,k}}{q_{u,k}}$, throughput ρ
- Set period length: $T_p = \text{lcm}\{q_{u,k}\}$
- During each period, send $n_u^{(k)} = \alpha_u^{(k)} \cdot T_{\text{period}}$ to each worker P_u
  $\Rightarrow$ periodic schedule with throughput ρ
- Initialization and clean-up phases
- Asymptotically optimal schedule (computes optimal number of tasks in time T, up to a constant independent of T)

Reconstructing an Optimal Schedule

- Solution of linear program: $\alpha_u^{(k)} = \frac{p_{u,k}}{q_{u,k}}$, throughput ρ
- Set period length: $T_p = \text{lcm}\{q_{u,k}\}$
- During each period, send $n_u^{(k)} = \alpha_u^{(k)} \cdot T_{\text{period}}$ to each worker P_u
  $\Rightarrow$ periodic schedule with throughput ρ
- Initialization and clean-up phases
- Asymptotically optimal schedule (computes optimal number of tasks in time T, up to a constant independent of T)

Reconstructing an Optimal Schedule

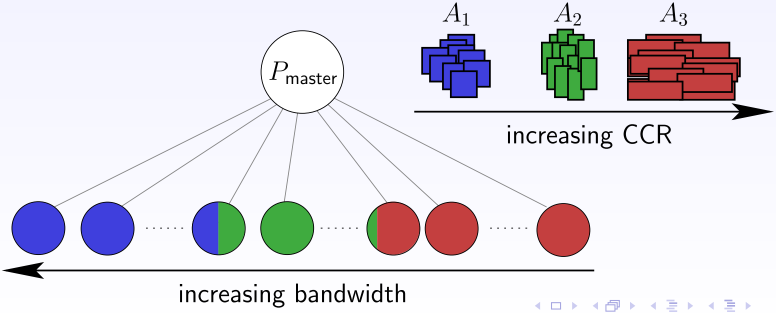
- Solution of linear program: $\alpha_u^{(k)} = \frac{p_{u,k}}{q_{u,k}}$, throughput ρ
- Set period length: $T_p = \text{lcm}\{q_{u,k}\}$
- During each period, send $n_u^{(k)} = \alpha_u^{(k)} \cdot T_{\text{period}}$ to each worker P_u $\Rightarrow$ periodic schedule with throughput ρ
- Initialization and clean-up phases
- Asymptotically optimal schedule (computes optimal number of tasks in time T, up to a constant independent of T)

Structure of the Optimal Solution

Theorem

- Sort the link by bandwidth so that $b_1 \geqslant b_2 \ldots \geqslant b_p$.
- Sort the applications by CCR so that $\frac{b^{(1)}}{c^{(1)}} \geqslant \frac{b^{(2)}}{c^{(2)}} \ldots \geqslant \frac{b^{(K)}}{c^{(K)}}$.

Then there exist indices $a_0 \leqslant a_1 \ldots \leqslant a_K$, $a_0 = 1$, $a_{k-1} \leqslant a_k$ for $1 \leqslant k \leqslant K$, $a_K \leqslant p$, such that only processors P_u, $u \in [a_{k-1}, a_k]$, execute tasks of type k in the optimal solution.



Adaptation to Tree Networks

- Linear Program can be extended
- Similarly reconstruction of periodic schedule
- No proof of a particular structure

Adaptation to Tree Networks

- Linear Program can be extended
- Similarly reconstruction of periodic schedule
- No proof of a particular structure

Adaptation to Tree Networks

- Linear Program can be extended
- Similarly reconstruction of periodic schedule
- No proof of a particular structure

Problems in previous solutions

- LP approach:
  - Centralized, needs all global information at master
  - Schedule period possibly huge
    $\longrightarrow$ difficult to adapt to load variation
  - Large memory requirement

Problems in previous solutions

- LP approach:
  - Centralized, needs all global information at master
  - Schedule period possibly huge
    $\longrightarrow$ difficult to adapt to load variation
  - Large memory requirement

Problems in previous solutions

- LP approach:
  - Centralized, needs all global information at master
  - Schedule period possibly huge
    $\longrightarrow$ difficult to adapt to load variation
  - Large memory requirement

Outline

Decentralized Heuristics

- General scheme for a decentralized heuristic:
  - Finite buffer (makes the problem NP hard)
  - *Demand-driven* algorithms
  - Local scheduler:

    Loop
        If there will be room in your buffer, request work from parent.
        Select which child to assign work to.
        Select the type of application that will be assigned.
        Get incoming requests from your local worker and children, if any.
        Move incoming tasks from your parent, if any, into your buffer.
        If you have a task and a request that match your choice **Then**
            Send the task to the chosen thread (when the send port is free)
        Else
            Wait for a request or a task
  - Use only *local* information

Heuristics – LP

- Centralized LP based (LP)
  - Solve linear program with global information
  - Give each node the $\alpha_u^{(k)}$ for its children and himself
  - Use a 1D load balancing mechanism with these ratios
    $\longrightarrow$ close to optimal throughput?

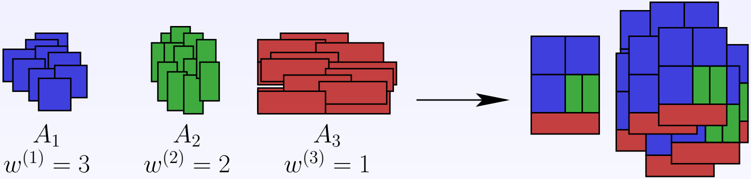
- First Come First Served (FCFS)
  - Each scheduler enforces a FCFS policy
  - Master ensures fairness using 1D load balancing mechanism

Heuristics – LP

- Centralized LP based (LP)
  - Solve linear program with global information
  - Give each node the $\alpha_u^{(k)}$ for its children and himself
  - Use a 1D load balancing mechanism with these ratios
    $\longrightarrow$ close to optimal throughput?
- First Come First Served (FCFS)
  - Each scheduler enforces a FCFS policy
  - Master ensures fairness using 1D load balancing mechanism

Heuristics – CGBC

- Coarse-Grain Bandwidth-Centric (CGBC)
  - Bandwidth-centric = optimal solution for a single application (send tasks to children communicating faster first)
  - Assemble different types of tasks into one macro-task:



  - Not expected to reach optimal throughput: slow links are used to transfer tasks with high CCR

Heuristics – PBC

- Parallel Bandwidth-Centric (PBC)
  - Superpose bandwidth-centric strategy for each application
  - On each worker, K independent schedulers
  - Fairness enforced by the master, distributing the tasks
  - Independent schedulers $\longrightarrow$ concurrent transfers
  - Limited capacity on outgoing port
    $\leadsto$ gives an (unfair) advantage to PBC (allows interruptible communications)

Heuristics – PBC

- Parallel Bandwidth-Centric (PBC)
  - Superpose bandwidth-centric strategy for each application
  - On each worker, K independent schedulers
  - Fairness enforced by the master, distributing the tasks
  - Independent schedulers $\rightarrow$ concurrent transfers
  - Limited capacity on outgoing port
    $\leadsto$ gives an (unfair) advantage to PBC (allows interruptible communications)

Heuristics – PBC

- Parallel Bandwidth-Centric (PBC)
  - Superpose bandwidth-centric strategy for each application
  - On each worker, K independent schedulers
  - Fairness enforced by the master, distributing the tasks
  - Independent schedulers $\rightarrow$ concurrent transfers
  - Limited capacity on outgoing port
    $\rightsquigarrow$ gives an (unfair) advantage to PBC (allows interruptible communications)

Heuristics – PBC

- Parallel Bandwidth-Centric (PBC)
  - Superpose bandwidth-centric strategy for each application
  - On each worker, K independent schedulers
  - Fairness enforced by the master, distributing the tasks
  - Independent schedulers $\rightarrow$ concurrent transfers
  - Limited capacity on outgoing port
    $\rightsquigarrow$ gives an (unfair) advantage to PBC (allows interruptible communications)

Heuristics – PBC

- Parallel Bandwidth-Centric (PBC)
  - Superpose bandwidth-centric strategy for each application
  - On each worker, K independent schedulers
  - Fairness enforced by the master, distributing the tasks
  - Independent schedulers $\rightarrow$ concurrent transfers
  - Limited capacity on outgoing port
    $\rightsquigarrow$ gives an (unfair) advantage to PBC (allows interruptible communications)

Heuristics – DATA-CENTRIC

- Data-centric scheduling (DATA-CENTRIC)
  - Decentralized heuristic
  - Try to convergence to the solution of LP
  - Intuition based on the structure of optimal solution for star networks
  - Start by scheduling only tasks with higher CCR, then periodically:
    - ★ substitute tasks of type A (high CCR) for tasks of type B (lower CCR)
    - ★ if unused bandwidth appears, send more tasks with high CCR
    - ★ if only tasks with high CCR are sent, lower this quantity to free bandwidth, in order to send other types of tasks
  - Needs information on neighbors
  - Some operations are decided on the master, then propagated along the tree

Heuristics – DATA-CENTRIC

- Data-centric scheduling (DATA-CENTRIC)
  - Decentralized heuristic
  - Try to convergence to the solution of LP
  - Intuition based on the structure of optimal solution for star networks
  - Start by scheduling only tasks with higher CCR, then periodically:
    - substitute tasks of type A (high CCR) for tasks of type B (lower CCR)
    - if unused bandwidth appears, send more tasks with high CCR
    - if only tasks with high CCR are sent, lower this quantity to free bandwidth, in order to send other types of tasks
  - Needs information on neighbors
  - Some operations are decided on the master, then propagated along the tree

Heuristics – DATA-CENTRIC

- Data-centric scheduling (DATA-CENTRIC)
  - Decentralized heuristic
  - Try to convergence to the solution of LP
  - Intuition based on the structure of optimal solution for star networks
  - Start by scheduling only tasks with higher CCR, then periodically:
    - ⋆ substitute tasks of type A (high CCR) for tasks of type B (lower CCR)
    - ⋆ if unused bandwidth appears, send more tasks with high CCR
    - ⋆ if only tasks with high CCR are sent, lower this quantity to free bandwidth, in order to send other types of tasks
  - Needs information on neighbors
  - Some operations are decided on the master, then propagated along the tree

Heuristics – DATA-CENTRIC

- Data-centric scheduling (DATA-CENTRIC)
  - Decentralized heuristic
  - Try to convergence to the solution of LP
  - Intuition based on the structure of optimal solution for star networks
  - Start by scheduling only tasks with higher CCR, then periodically:
    - substitute tasks of type A (high CCR) for tasks of type B (lower CCR)
    - if unused bandwidth appears, send more tasks with high CCR
    - if only tasks with high CCR are sent, lower this quantity to free bandwidth, in order to send other types of tasks
  - Needs information on neighbors
  - Some operations are decided on the master, then propagated along the tree

Heuristics – DATA-CENTRIC

- Data-centric scheduling (DATA-CENTRIC)
  - Decentralized heuristic
  - Try to convergence to the solution of LP
  - Intuition based on the structure of optimal solution for star networks
  - Start by scheduling only tasks with higher CCR, then periodically:
    - substitute tasks of type A (high CCR) for tasks of type B (lower CCR)
    - if unused bandwidth appears, send more tasks with high CCR
    - if only tasks with high CCR are sent, lower this quantity to free bandwidth, in order to send other types of tasks
  - Needs information on neighbors
  - Some operations are decided on the master, then propagated along the tree

Outline

Methodology

- How to measure fair-throughput ?
  - Concentrate on phase where all applications simultaneously run
    $\rightarrow T =$ first time s.t. all tasks of some application are terminated
  - Ignore initialization and termination phases
  - Set time-interval: $[0.1 \times T\ ;\ \ 0.9 \times T]$
  - Compute achieved throughput for each application on this interval
- Platform generation
  - 150 random platforms generated, preferring wide trees
  - Links and processors characteristics based on measured values
  - Buffer of size 10 tasks (of any type)
- Application generation
  - CCR chosen between 0.001 (matrix multiplication) and 4.6 (matrix addition)
- Heuristic implementation
  - Distributed implementation using SimGrid,
  - Link and processor capacities measured within SimGrid

Methodology

- How to measure fair-throughput ?
  - Concentrate on phase where all applications simultaneously run $\rightarrow T =$ first time s.t. all tasks of some application are terminated
  - Ignore initialization and termination phases
  - Set time-interval: $[0.1 \times T \; ; \; 0.9 \times T]$
  - Compute achieved throughput for each application on this interval
- Platform generation
  - 150 random platforms generated, preferring wide trees
  - Links and processors characteristics based on measured values
  - Buffer of size 10 tasks (of any type)
- Application generation
  - CCR chosen between 0.001 (matrix multiplication) and 4.6 (matrix addition)
- Heuristic implementation
  - Distributed implementation using SimGrid,
  - Link and processor capacities measured within SimGrid

Methodology

- How to measure fair-throughput ?
  - Concentrate on phase where all applications simultaneously run
    $\rightarrow T =$ first time s.t. all tasks of some application are terminated
  - Ignore initialization and termination phases
  - Set time-interval: $[0.1 \times T\ ;\ \ 0.9 \times T]$
  - Compute achieved throughput for each application on this interval
- Platform generation
  - 150 random platforms generated, preferring wide trees
  - Links and processors characteristics based on measured values
  - Buffer of size 10 tasks (of any type)
- Application generation
  - CCR chosen between 0.001 (matrix multiplication) and 4.6 (matrix addition)
- Heuristic implementation
  - Distributed implementation using SimGrid,
  - Link and processor capacities measured within SimGrid

Methodology

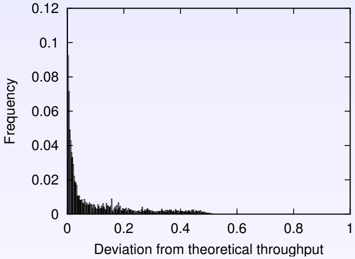
- How to measure fair-throughput ?
  - Concentrate on phase where all applications simultaneously run
    $\rightarrow T =$ first time s.t. all tasks of some application are terminated
  - Ignore initialization and termination phases
  - Set time-interval: $[0.1 \times T \ ; \ \ 0.9 \times T]$
  - Compute achieved throughput for each application on this interval
- Platform generation
  - 150 random platforms generated, preferring wide trees
  - Links and processors characteristics based on measured values
  - Buffer of size 10 tasks (of any type)
- Application generation
  - CCR chosen between 0.001 (matrix multiplication) and 4.6 (matrix addition)
- Heuristic implementation
  - Distributed implementation using SimGrid,
  - Link and processor capacities measured within SimGrid

Theoretical v/ Experimental Throughput

- LP, CGBC: possible to compute expected (theoretical) throughput
-

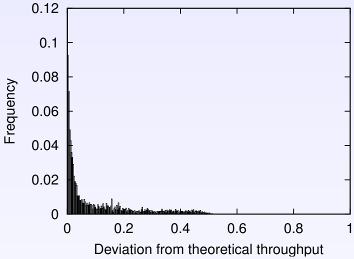
average deviation = 9.4%
- Increase buffer size from 10 to 200 → average deviation = 0.3%
- In the following, LP = basis for comparison
- Compute $\log \frac{\text{performance of H}}{\text{performance of LP}}$ for each heuristic H, on each platform
- Plot distribution

Theoretical v/ Experimental Throughput

- LP, CGBC: possible to compute expected (theoretical) throughput
- 

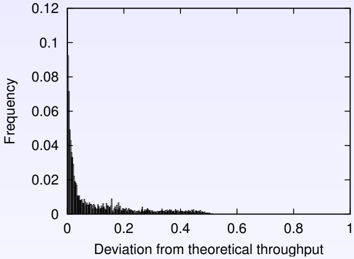
  average deviation = 9.4%
- Increase buffer size from 10 to 200 → average deviation = 0.3%
- In the following, LP = basis for comparison
- Compute $\log \frac{\text{performance of H}}{\text{performance of LP}}$ for each heuristic H, on each platform
- Plot distribution

Theoretical v/ Experimental Throughput

- LP, CGBC: possible to compute expected (theoretical) throughput
- 

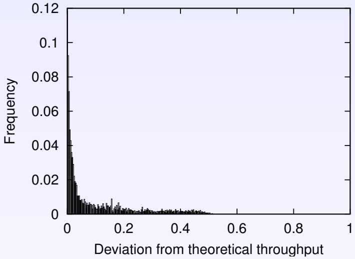
  average deviation = 9.4%
- Increase buffer size from 10 to 200 $\longrightarrow$ average deviation = 0.3%
- In the following, LP = basis for comparison
- Compute $\log \frac{\text{performance of H}}{\text{performance of LP}}$ for each heuristic H, on each platform
- Plot distribution

Theoretical v/ Experimental Throughput

- LP, CGBC: possible to compute expected (theoretical) throughput
- 

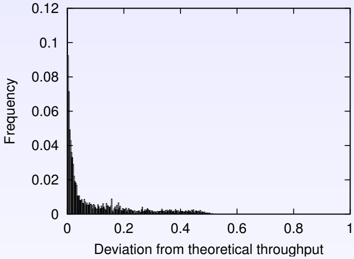
average deviation = 9.4%
- Increase buffer size from 10 to 200 → average deviation = 0.3%
- In the following, LP = basis for comparison
- Compute $\log \frac{\text{performance of H}}{\text{performance of LP}}$ for each heuristic H, on each platform
- Plot distribution

Theoretical v/ Experimental Throughput

- LP, CGBC: possible to compute expected (theoretical) throughput
- 

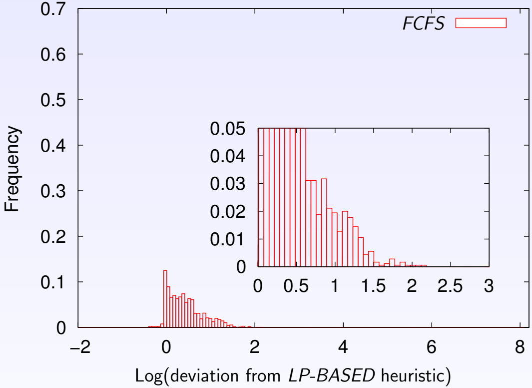
  average deviation = 9.4%
- Increase buffer size from 10 to 200 $\longrightarrow$ average deviation = 0.3%
- In the following, LP = basis for comparison
- Compute $\log \frac{\text{performance of H}}{\text{performance of LP}}$
  for each heuristic H, on each platform
- Plot distribution

Theoretical v/ Experimental Throughput

- LP, CGBC: possible to compute expected (theoretical) throughput
- 

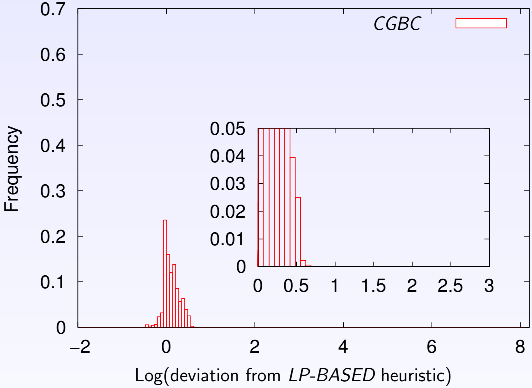
  average deviation = 9.4%
- Increase buffer size from 10 to 200 $\rightarrow$ average deviation = 0.3%
- In the following, LP = basis for comparison
- Compute $\log \frac{\text{performance of H}}{\text{performance of LP}}$ for each heuristic H, on each platform
- Plot distribution

Performance of FCFS



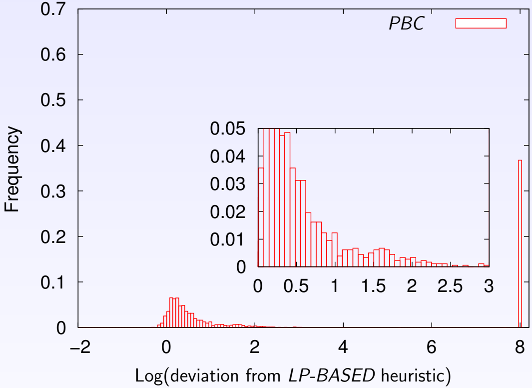
- Geometrical average: FCFS is 1.56 times worse than LP
- Worst case: 8 times worse

Performance of CGBC



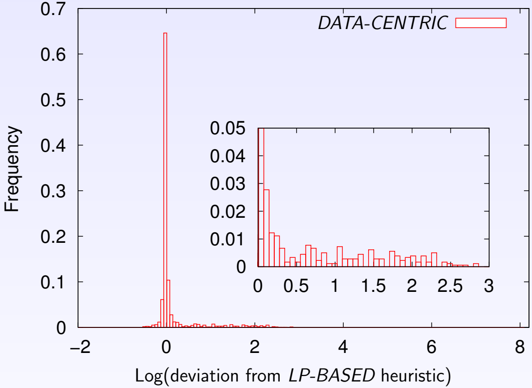
- Geometrical average: CGBC is 1.15 times worse than LP
- Worst case: 2 times worse

Performance of PBC



- In 35% of the cases: one application is totally unfavored, its throughput is close to 0.

Performance of DATA-CENTRIC



- Geometrical average: DATA-CENTRIC is 1.16 worse than LP
- Few instances with very bad solution
- On most platforms, very good solution
- Hard to know why it performs badly in few cases

Outline

Conclusion

- Centralized algorithm computes optimal solution with global information
- Nice characterization of optimal solution on single-level trees
- Design distributed heuristics to deal with practical settings of clusters and grids (distributed information, variability, limited memory)
- Evaluation of these heuristics through extensive simulations
- Good performance of sophisticated heuristics compared to the optimal scheduling

Perspectives

- Adapt decentralized MultiCommodity Flow algorithm (Awerbuch & Leighton) to our problem
  - Decentralized approach to compute optimal throughput
  - Slow convergence speed
- Consider other kinds of fairness such as proportional fairness:
  - Reasonable (close to the behavior of TCP)
  - Easy to enforce with distributed algorithms