Offline and Online Master-Worker Scheduling of Concurrent Bags-of-Tasks on Heterogeneous Platforms

Loris MARCHAL,

joint work with Anne BENOIT, Jean-François PINEAU, Yves ROBERT and Frédéric VIVIEN

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

Scheduling in Aussois,
May, 18 - 21 2008.

Object of the Study

- Bags-of-tasks application
  - independent tasks
  - large number of similar tasks
  - models embarrassingly parallel applications
  - argues for the use of wide distributed platforms
- Online scheduling
  - applications arrive at different times (release dates)
  - no knowledge on the future
  - no global makespan, try to lower the suffering of each user

Object of the Study

- Bags-of-tasks application
  - independent tasks
  - large number of similar tasks
  - models embarrassingly parallel applications
  - argues for the use of wide distributed platforms

- Online scheduling
  - applications arrive at different times (release dates)
  - no knowledge on the future
  - no global makespan, try to lower the suffering of each user

Building on our previous results

- Large number of tasks ⇒ steady-state scheduling
  - designed for large applications
  - suited for heterogeneous platforms, multiple applications

(*Centralized versus distributed schedulers for multiple bag-of-task applications*, IPDPS'06)

  - optimal platform utilization: throughput maximization
  - neglect transient phases (initialization/clean-up)

- Online scheduling ⇒ maximum stretch minimization
  - other metrics not suited

(*Minimizing the stretch when scheduling flows of biological requests*, SPAA '06)

  - stretch is a kind of *price for sharing resources*
  - minimize the maximum stretch among applications: give a guarantee on each application slowdown

NB: maximize throughput and minimize max-stretch could seem contradictory

Building on our previous results

- Large number of tasks ⇒ steady-state scheduling
  - designed for large applications
  - suited for heterogeneous platforms, multiple applications

(*Centralized versus distributed schedulers for multiple bag-of-task applications*, IPDPS'06)
  - optimal platform utilization: throughput maximization
  - neglect transient phases (initialization/clean-up)

- Online scheduling ⇒ maximum stretch minimization
  - other metrics not suited

(*Minimizing the stretch when scheduling flows of biological requests*, SPAA '06)
  - stretch is a kind of *price for sharing resources*
  - minimize the maximum stretch among applications:
    give a guarantee on each application slowdown

NB: maximize throughput and minimize max-stretch could seem contradictory

Building on our previous results

- Large number of tasks ⇒ steady-state scheduling
  - designed for large applications
  - suited for heterogeneous platforms, multiple applications

(*Centralized versus distributed schedulers for multiple bag-of-task applications*, IPDPS'06)

  - optimal platform utilization: throughput maximization
  - neglect transient phases (initialization/clean-up)

- Online scheduling ⇒ maximum stretch minimization
  - other metrics not suited

(*Minimizing the stretch when scheduling flows of biological requests*, SPAA '06)

  - stretch is a kind of *price for sharing resources*
  - minimize the maximum stretch among applications: give a guarantee on each application slowdown

NB: maximize throughput and minimize max-stretch could seem contradictory

Simple idea to bring things together

- Suppose we want to reach the maximum stretch $\mathcal{S}$
- For a given application, we can compute its makespan "if it was alone": MS
- This gives a deadline:

$$\text{deadline} = \text{release date} + \mathcal{S} \times MS$$

- Each application has now a release date and a deadline.
- Dates define intervals...
  where we can apply steady-state relaxation!

Simple idea to bring things together

- Suppose we want to reach the maximum stretch $\mathcal{S}$
- For a given application, we can compute its makespan "if it was alone": MS
- This gives a deadline:

$$\text{deadline} = \text{release date} + \mathcal{S} \times MS$$

- Each application has now a release date and a deadline.
- Dates define intervals...
  where we can apply steady-state relaxation!

Simple idea to bring things together

- Suppose we want to reach the maximum stretch $\mathcal{S}$
- For a given application, we can compute its makespan "if it was alone": MS
- This gives a deadline:

$$\text{deadline} = \text{release date} + \mathcal{S} \times MS$$

- Each application has now a release date and a deadline.
- Dates define intervals...

  where we can apply steady-state relaxation!

Simple idea to bring things together

- Suppose we want to reach the maximum stretch $\mathcal{S}$
- For a given application, we can compute its makespan "if it was alone": MS
- This gives a deadline:

$$\text{deadline} = \text{release date} + \mathcal{S} \times MS$$

- Each application has now a release date and a deadline.
- Dates define intervals...

  where we can apply steady-state relaxation!

Simple idea to bring things together

- Suppose we want to reach the maximum stretch $\mathcal{S}$
- For a given application, we can compute its makespan "if it was alone": MS
- This gives a deadline:

$$\text{deadline} = \text{release date} + \mathcal{S} \times MS$$

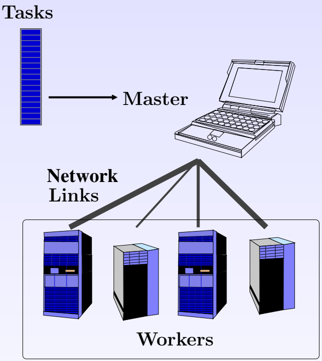
- Each application has now a release date and a deadline.
- Dates define intervals...
  where we can apply steady-state relaxation!

Outline

Outline

Single bag-of-task application – context

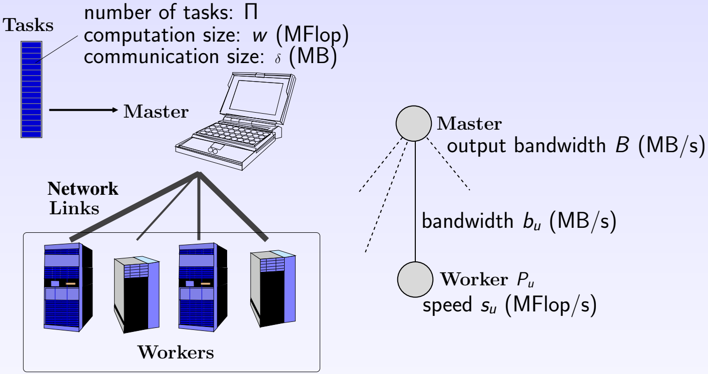
- Master-Slave platform (heterogeneous):



- Bunch of identical tasks
- Computing optimal makespan: already difficult problem
- Steady-state relaxation to get a lower bound

Single bag-of-task application – context

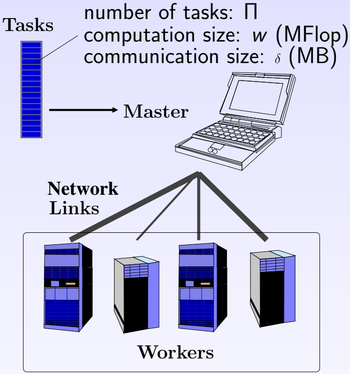
- Master-Slave platform (heterogeneous):

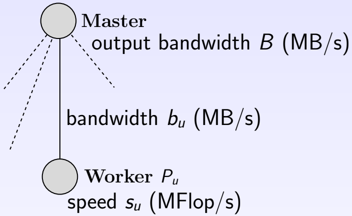


- Bunch of identical tasks
- Computing optimal makespan: already difficult problem
- Steady-state relaxation to get a lower bound

Single bag-of-task application – context

- Master-Slave platform (heterogeneous):

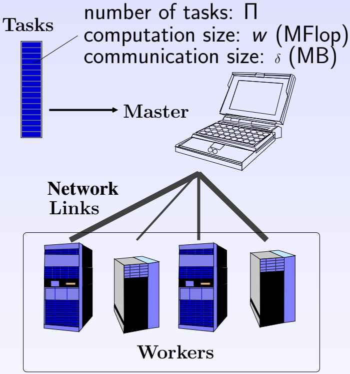


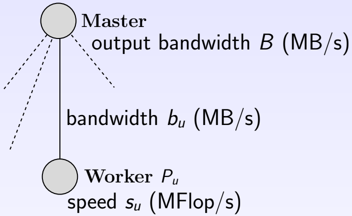


- Bunch of identical tasks
- Computing optimal makespan: already difficult problem
- Steady-state relaxation to get a lower bound

Single bag-of-task application – context

- Master-Slave platform (heterogeneous):

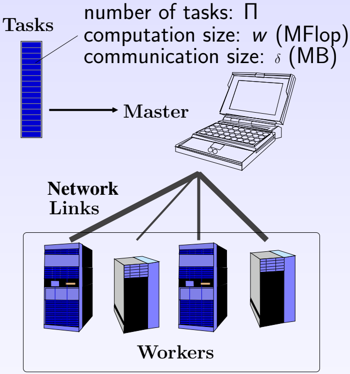


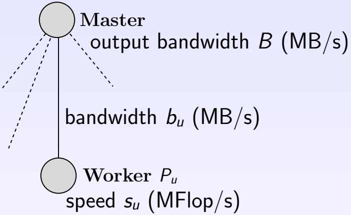


- Bunch of identical tasks
- Computing optimal makespan: already difficult problem
- Steady-state relaxation to get a lower bound

Single bag-of-task application – context

- Master-Slave platform (heterogeneous):





- Bunch of identical tasks
- Computing optimal makespan: already difficult problem
- Steady-state relaxation to get a lower bound

Single bag-of-task application – steady-state

Motivations:

- Assume the number of tasks is huge
- Forget about makespan (meaningless)
- Concentrate on throughput (fluid framework)

How it works:

- Consider average values:
  "master sends 5.3 tasks per second to worker 3"
- Write constraints on these variables
- Optimize total throughput under these constraints
  (with the help of linear programming)
- Reconstruct near-optimal schedule from average values
  (we skip this step for now)

Single bag-of-task application – steady-state

Motivations:

- Assume the number of tasks is huge
- Forget about makespan (meaningless)
- Concentrate on throughput (fluid framework)

How it works:

- Consider average values:
  "master sends 5.3 tasks per second to worker 3"
- Write constraints on these variables
- Optimize total throughput under these constraints
  (with the help of linear programming)
- Reconstruct near-optimal schedule from average values
  (we skip this step for now)

Single bag-of-task application – steady-state

Motivations:

- Assume the number of tasks is huge
- Forget about makespan (meaningless)
- Concentrate on throughput (fluid framework)

How it works:

- Consider average values:
  "master sends 5.3 tasks per second to worker 3"
- Write constraints on these variables
- Optimize total throughput under these constraints
  (with the help of linear programming)
- Reconstruct near-optimal schedule from average values
  (we skip this step for now)

Single bag-of-task application – linear program

$$\begin{cases} \text{MAXIMIZE } \rho = \sum_{u=1}^{p} \rho_u \\ \text{SUBJECT TO} \\ \forall P_u, \quad \rho_u \frac{w}{s_u} \leq 1 \\ \forall P_u, \quad \rho_u \frac{\delta}{b_u} \leq 1 \\ \sum_{u=1}^{p} \rho_u \frac{\delta}{\mathcal{B}} \leq 1 \end{cases}$$

ρ_u: throughput of worker P_u
ρ: Total throughput

Analytical solution

$$\rho = \min \left\{ \frac{\mathcal{B}}{\delta}, \sum_{u=1}^{p} \min \left\{ \frac{s_u}{w}, \frac{b_u}{w} \right\} \right\}$$

Estimated makespan (lower bound):

$$MS = \frac{\text{number of tasks}}{\text{optimal throughput}} = \frac{\Pi}{\rho}$$

Single bag-of-task application – linear program

$$\begin{cases} \textsc{Maximize } \rho = \sum_{u=1}^{p} \rho_u \\ \text{SUBJECT TO} \\ \forall P_u, \quad \rho_u \dfrac{w}{s_u} \leq 1 \\ \forall P_u, \quad \rho_u \dfrac{\delta}{b_u} \leq 1 \\ \sum_{u=1}^{p} \rho_u \dfrac{\delta}{\mathcal{B}} \leq 1 \end{cases}$$

ρ_u: throughput of worker P_u
ρ: Total throughput

Analytical solution

$$\rho = \min \left\{ \frac{\mathcal{B}}{\delta}, \sum_{u=1}^{p} \min \left\{ \frac{s_u}{w}, \frac{b_u}{w} \right\} \right\}$$

Estimated makespan (lower bound):

$$MS = \frac{\text{number of tasks}}{\text{optimal throughput}} = \frac{\Pi}{\rho}$$

Single bag-of-task application – linear program

$$\begin{cases} \textsc{Maximize } \rho = \sum_{u=1}^{p} \rho_u \\ \textsc{subject to} \\ \forall P_u, \quad \rho_u \dfrac{w}{s_u} \leq 1 \\ \forall P_u, \quad \rho_u \dfrac{\delta}{b_u} \leq 1 \\ \sum_{u=1}^{p} \rho_u \dfrac{\delta}{\mathcal{B}} \leq 1 \end{cases}$$

ρ_u: throughput of worker P_u
ρ: Total throughput

Analytical solution

$$\rho = \min \left\{ \frac{\mathcal{B}}{\delta}, \sum_{u=1}^{p} \min \left\{ \frac{s_u}{w}, \frac{b_u}{w} \right\} \right\}$$

Estimated makespan (lower bound):

$$MS = \frac{\text{number of tasks}}{\text{optimal throughput}} = \frac{\Pi}{\rho}$$

Outline

Offline multi-application – framework

For each application k (task of sizes $w^{(k)}$, $\delta^{(k)}$), we have:

- a release date
- an estimated makespan $MS^{*(k)}$ (lower bound)

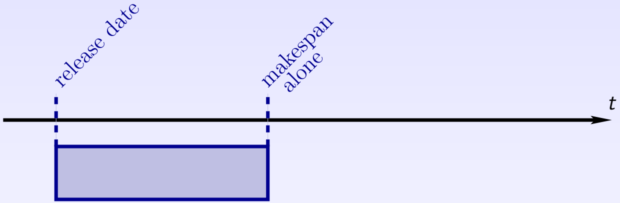
We try to reach stretch $\mathcal{S}$:

- deadline:

$$\text{deadline}^{(k)} = \text{release date}^{(k)} + \mathcal{S} \times MS^{*(k)}$$

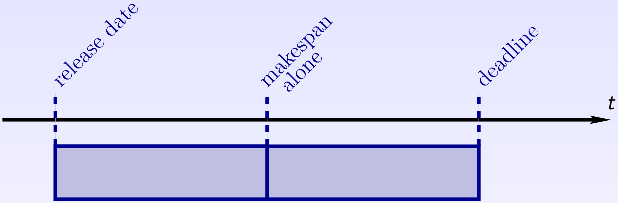
Time-intervals for target stretch

If we try to reach stretch $\mathcal{S} = 2$:



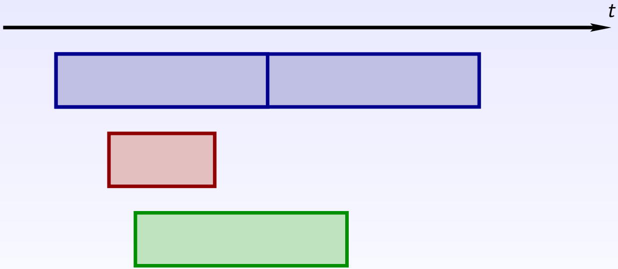
Time-intervals for target stretch

If we try to reach stretch $\mathcal{S} = 2$:



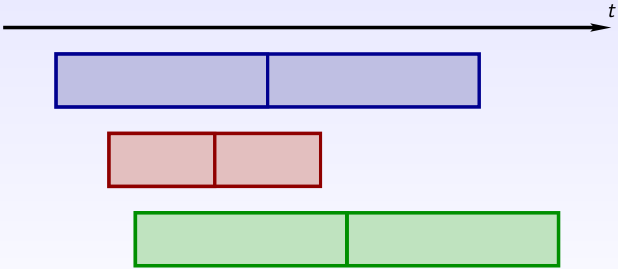
Time-intervals for target stretch

If we try to reach stretch $\mathcal{S} = 2$:



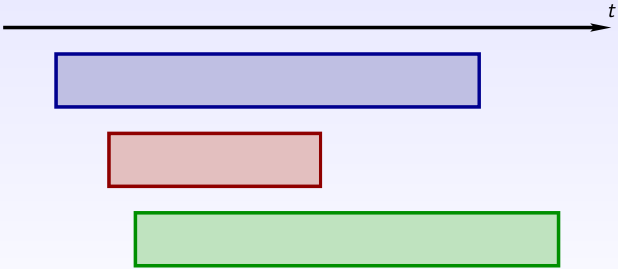
Time-intervals for target stretch

If we try to reach stretch $\mathcal{S} = 2$:



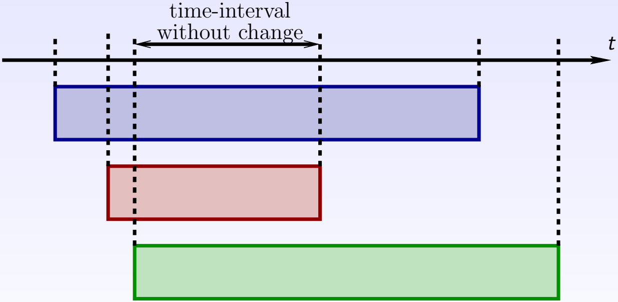
Time-intervals for target stretch

If we try to reach stretch $\mathcal{S} = 2$:



Time-intervals for target stretch

If we try to reach stretch $\mathcal{S} = 2$:



Resolution for a target stretch $\mathcal{S}$

New variables:

- communication throughput $\rho^{(k)}_{M\to u}(t_j, t_{j+1})$
- computation throughput $\rho^{(k)}_{u}(t_j, t_{j+1})$
- state of buffers: $B^{(k)}_u(t_j)$
  (number of non-executed tasks at time t_j)

New constraints:

- Complex (but straightforward) conservation laws between throughputs and buffer state
- Assert that all tasks of an application are treated.
- Resource limitations

Set of linear constraints, defining a convex $K(\mathcal{S})$. details

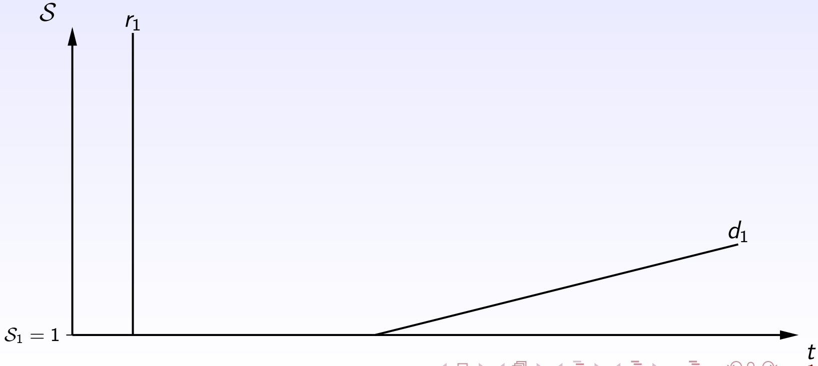
$$K(\mathcal{S}) \text{ non-empty} \Leftrightarrow \mathcal{S} \text{ feasible}$$

Binary search of optimal stretch

We have a toolbox to know if a given stretch is feasible. Search of the optimal (minimum) stretch:

- Basic binary search (with precision ϵ), or
- Involved search among stretch-intervals:

$$d^{(k)}(\mathcal{S}) = r^{(k)} + \mathcal{S} \times MS^{*(k)}.$$

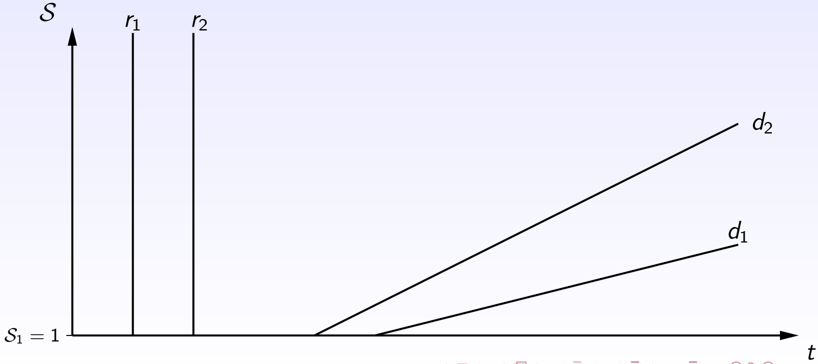


Binary search of optimal stretch

We have a toolbox to know if a given stretch is feasible. Search of the optimal (minimum) stretch:

- Basic binary search (with precision ϵ), or
- Involved search among stretch-intervals:

$$d^{(k)}(\mathcal{S}) = r^{(k)} + \mathcal{S} \times MS^{*(k)}.$$

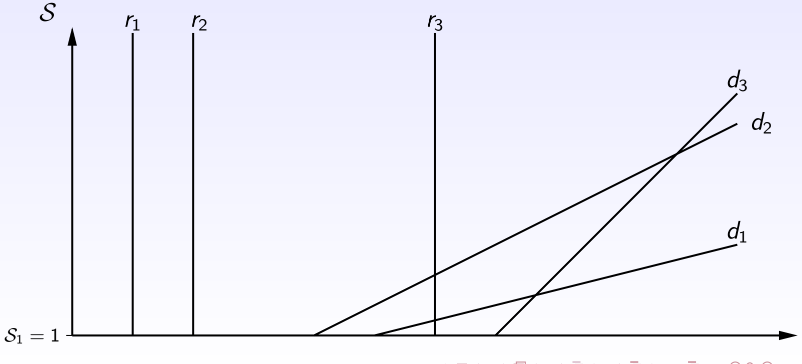


Binary search of optimal stretch

We have a toolbox to know if a given stretch is feasible. Search of the optimal (minimum) stretch:

- Basic binary search (with precision ϵ), or
- Involved search among stretch-intervals:

$$d^{(k)}(\mathcal{S}) = r^{(k)} + \mathcal{S} \times MS^{*(k)}.$$

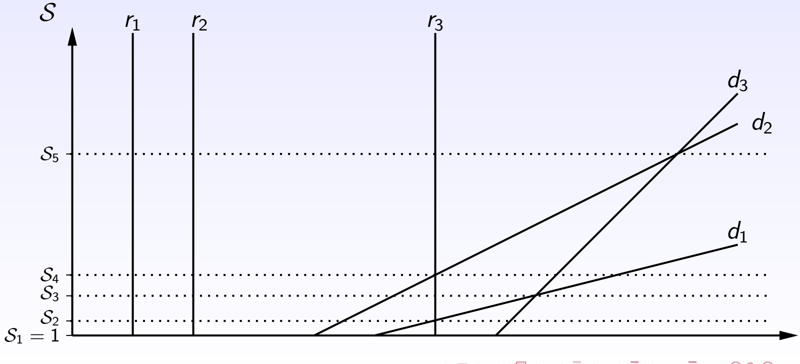


Binary search of optimal stretch

We have a toolbox to know if a given stretch is feasible. Search of the optimal (minimum) stretch:

- Basic binary search (with precision ϵ), or
- Involved search among stretch-intervals:

$$d^{(k)}(\mathcal{S}) = r^{(k)} + \mathcal{S} \times MS^{*(k)}.$$

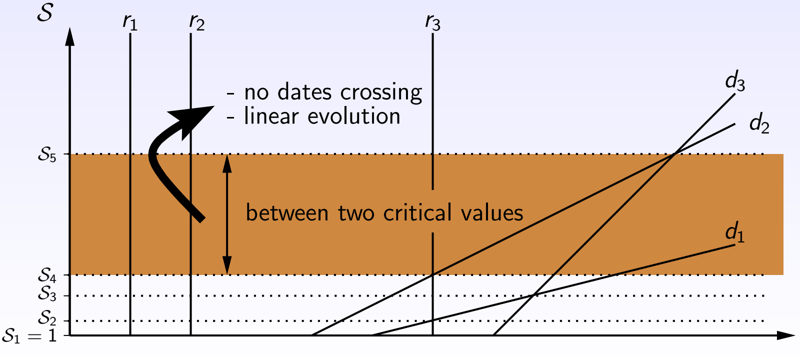


Binary search of optimal stretch

We have a toolbox to know if a given stretch is feasible. Search of the optimal (minimum) stretch:

- Basic binary search (with precision ϵ), or
- Involved search among stretch-intervals:

$$d^{(k)}(\mathcal{S}) = r^{(k)} + \mathcal{S} \times MS^{*(k)}.$$



Binary search between stretch-interval

- Consider a stretch-interval between two critical values $[\mathcal{S}_a ; \mathcal{S}_b]$
- Deadlines have a linear evolution
- Everything is linear ? Not really:
  when computing what receives a buffer during a time-interval:

  $T_{\text{end}}, T_{\text{start}}$: linear function in $\mathcal{S}$
  $\rightsquigarrow$ quadratic constraints ☹
- Switch from *throughput* to *amount* variables:

$$
\begin{aligned}
A^{(k)}_{M\to u}(t_j, t_{j+1}) &= \rho^{(k)}_{M\to u}(t_j, t_{j+1}) \times (t_{j+1} - t_j) \\
A^{(k)}_{u}(t_j, t_{j+1}) &= \rho^{(k)}_{u}(t_j, t_{j+1}) \times (t_{j+1} - t_j)
\end{aligned}
$$

- All the constraints are once again linear ☺

Binary search between stretch-interval

- Consider a stretch-interval between two critical values $[\mathcal{S}_a; \mathcal{S}_b]$
- Deadlines have a linear evolution
- Everything is linear ? Not really:

$$\rho^{(k)}_{M \to u}(t_j, t_{j+1}) \times (T_{\text{end}} - T_{\text{start}})$$

T_{end}, T_{start}: linear function in $\mathcal{S}$
$\rightsquigarrow$ quadratic constraints ☹

- Switch from *throughput* to *amount* variables:

$$\begin{aligned} A^{(k)}_{M \to u}(t_j, t_{j+1}) &= \rho^{(k)}_{M \to u}(t_j, t_{j+1}) \times (t_{j+1} - t_j) \\ A^{(k)}_{u}(t_j, t_{j+1}) &= \rho^{(k)}_{u}(t_j, t_{j+1}) \times (t_{j+1} - t_j) \end{aligned}$$

- All the constraints are once again linear ☺

Binary search between stretch-interval

- Consider a stretch-interval between two critical values $[\mathcal{S}_a;\mathcal{S}_b]$
- Deadlines have a linear evolution
- Everything is linear ? Not really:
  when computing what receives a buffer during a time-interval:

$$\rho^{(k)}_{M\to u}(t_j,t_{j+1}) \times (T_{\text{end}} - T_{\text{start}})$$

T_{end}, T_{start}: linear function in $\mathcal{S}$
$\rightsquigarrow$ quadratic constraints ☹

- Switch from *throughput* to *amount* variables:

$$\begin{array}{rcl} A^{(k)}_{M\to u}(t_j,t_{j+1}) & = & \rho^{(k)}_{M\to u}(t_j,t_{j+1}) \times (t_{j+1}-t_j) \\ A^{(k)}_{u}(t_j,t_{j+1}) & = & \rho^{(k)}_{u}(t_j,t_{j+1}) \times (t_{j+1}-t_j) \end{array}$$

- All the constraints are once again linear ☺ ▸ details

Binary search between stretch-interval

- Consider a stretch-interval between two critical values $[\mathcal{S}_a; \mathcal{S}_b]$
- Deadlines have a linear evolution
- Everything is linear ? Not really:
  when computing what receives a buffer during a time-interval:

$$\rho^{(k)}_{M \to u}(t_j, t_{j+1}) \times (T_{\text{end}} - T_{\text{start}})$$

$T_{\text{end}}, T_{\text{start}}$: linear function in $\mathcal{S}$
$\rightsquigarrow$ quadratic constraints ☹

- Switch from *throughput* to *amount* variables:

$$\begin{aligned} A^{(k)}_{M \to u}(t_j, t_{j+1}) &= \rho^{(k)}_{M \to u}(t_j, t_{j+1}) \times (t_{j+1} - t_j) \\ A^{(k)}_{u}(t_j, t_{j+1}) &= \rho^{(k)}_{u}(t_j, t_{j+1}) \times (t_{j+1} - t_j) \end{aligned}$$

- All the constraints are once again linear ☺ details

Binary search between stretch-interval

- Consider a stretch-interval between two critical values $[\mathcal{S}_a; \mathcal{S}_b]$
- Deadlines have a linear evolution
- Everything is linear ? Not really:
  when computing what receives a buffer during a time-interval:

$$\rho^{(k)}_{M \to u}(t_j, t_{j+1}) \times (T_{\text{end}} - T_{\text{start}})$$

  $T_{\text{end}}, T_{\text{start}}$: linear function in $\mathcal{S}$
  $\rightsquigarrow$ quadratic constraints ☹
- Switch from *throughput* to *amount* variables:

$$\begin{aligned} A^{(k)}_{M \to u}(t_j, t_{j+1}) &= \rho^{(k)}_{M \to u}(t_j, t_{j+1}) \times (t_{j+1} - t_j) \\ A^{(k)}_{u}(t_j, t_{j+1}) &= \rho^{(k)}_{u}(t_j, t_{j+1}) \times (t_{j+1} - t_j) \end{aligned}$$

- All the constraints are once again linear ☺ details

Binary search between stretch-interval

- Consider a stretch-interval between two critical values $[\mathcal{S}_a; \mathcal{S}_b]$
- Deadlines have a linear evolution
- Everything is linear ? Not really:
  when computing what receives a buffer during a time-interval:

  $$\rho^{(k)}_{M \to u}(t_j, t_{j+1}) \times (T_{\text{end}} - T_{\text{start}})$$

  $T_{\text{end}}, T_{\text{start}}$: linear function in $\mathcal{S}$
  $\rightsquigarrow$ quadratic constraints ☹
- Switch from *throughput* to *amount* variables:

  $$\begin{array}{rcl} A^{(k)}_{M \to u}(t_j, t_{j+1}) & = & \rho^{(k)}_{M \to u}(t_j, t_{j+1}) \times (t_{j+1} - t_j) \\ A^{(k)}_{u}(t_j, t_{j+1}) & = & \rho^{(k)}_{u}(t_j, t_{j+1}) \times (t_{j+1} - t_j) \end{array}$$

- All the constraints are once again linear ☺ ▸ details

Outline

Discussion on models

- Which communication/computation model have we been using from the beginning ?
- My favorite over-classical one-port model ?
  (a processor sends/receives one message at a time, and can overlap the communications by computations)
- No! no schedule reconstructed from the linear programs ☹
- Solution of a linear program : fluid throughput $\rho_u^{(k)}$, assumes
  - time-sharing for communication and computation
  - "Synchronous Start" for communication and computation
- Nice model for scheduling, but far from reality:
  - No data dependency (!)
  - Concurrent applications
  - Perfect time-sharing for computation and communication (!)
- We have to come back to the "reality"

Discussion on models

- Which communication/computation model have we been using from the beginning ?
- My favorite over-classical one-port model ?
  (a processor sends/receives one message at a time, and can overlap the communications by computations)
- No! no schedule reconstructed from the linear programs ☹
- Solution of a linear program : fluid throughput $\rho_u^{(k)}$, assumes
  - time-sharing for communication and computation
  - "Synchronous Start" for communication and computation
- Nice model for scheduling, but far from reality:
  - No data dependency (!)
  - Concurrent applications
  - Perfect time-sharing for computation and communication (!)
- We have to come back to the "reality"

Discussion on models

- Which communication/computation model have we been using from the beginning ?
- My favorite over-classical one-port model ?
  (a processor sends/receives one message at a time, and can overlap the communications by computations)
- No! no schedule reconstructed from the linear programs ☹
- Solution of a linear program : fluid throughput $\rho_u^{(k)}$, assumes
  - time-sharing for communication and computation
  - "Synchronous Start" for communication and computation
- Nice model for scheduling, but far from reality:
  - No data dependency (!)
  - Concurrent applications
  - Perfect time-sharing for computation and communication (!)
- We have to come back to the "reality"

Discussion on models

- Which communication/computation model have we been using from the beginning ?
- My favorite over-classical one-port model ?
  (a processor sends/receives one message at a time, and can overlap the communications by computations)
- No! no schedule reconstructed from the linear programs ☹
- Solution of a linear program : fluid throughput $\rho_u^{(k)}$, assumes
  - time-sharing for communication and computation
  - "Synchronous Start" for communication and computation
- Nice model for scheduling, but far from reality:
  - No data dependency (!)
  - Concurrent applications
  - Perfect time-sharing for computation and communication (!)
- We have to come back to the "reality"

Discussion on models

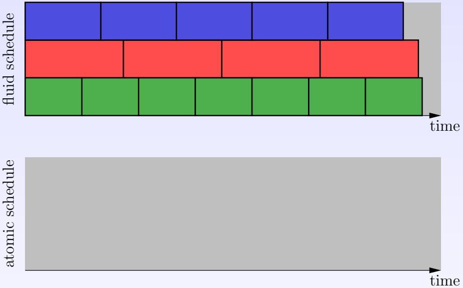
- Which communication/computation model have we been using from the beginning ?
- My favorite over-classical one-port model ?
  (a processor sends/receives one message at a time, and can overlap the communications by computations)
- No! no schedule reconstructed from the linear programs ☹
- Solution of a linear program : fluid throughput $\rho_u^{(k)}$, assumes
  - time-sharing for communication and computation
  - "Synchronous Start" for communication and computation
- Nice model for scheduling, but far from reality:
  - No data dependency (!)
  - Concurrent applications
  - Perfect time-sharing for computation and communication (!)
- We have to come back to the "reality"

Discussion on models

- Which communication/computation model have we been using from the beginning ?
- My favorite over-classical one-port model ?
  (a processor sends/receives one message at a time, and can overlap the communications by computations)
- No! no schedule reconstructed from the linear programs ☹
- Solution of a linear program : fluid throughput $\rho_u^{(k)}$, assumes
  - time-sharing for communication and computation
  - "Synchronous Start" for communication and computation
- Nice model for scheduling, but far from reality:
  - No data dependency (!)
  - Concurrent applications
  - Perfect time-sharing for computation and communication (!)
- We have to come back to the "reality"

One-dimensional load-balancing

- General fluid schedule with rate α_k for application k
- task of application k takes time t_k at full speed



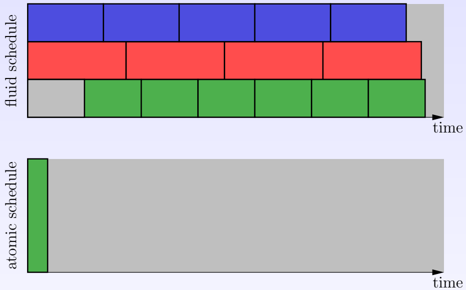
At each step, choose application which minimize

$$(n_k + 1) \times \frac{t_k}{\alpha_k}$$

n_k: number of task from application k already scheduled

One-dimensional load-balancing

- General fluid schedule with rate α_k for application k
- task of application k takes time t_k at full speed



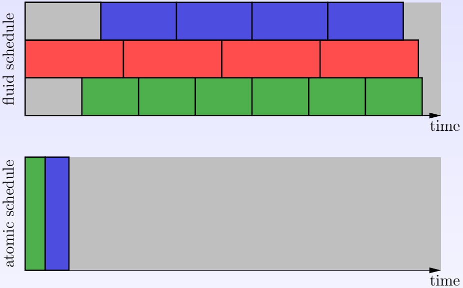
At each step, choose application which minimize

$$(n_k + 1) \times \frac{t_k}{\alpha_k}$$

n_k: number of task from application k already scheduled

One-dimensional load-balancing

- General fluid schedule with rate α_k for application k
- task of application k takes time t_k at full speed



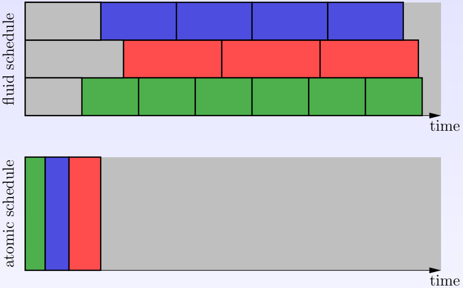
At each step, choose application which minimize

$$(n_k + 1) \times \frac{t_k}{\alpha_k}$$

n_k: number of task from application k already scheduled

One-dimensional load-balancing

- General fluid schedule with rate α_k for application k
- task of application k takes time t_k at full speed



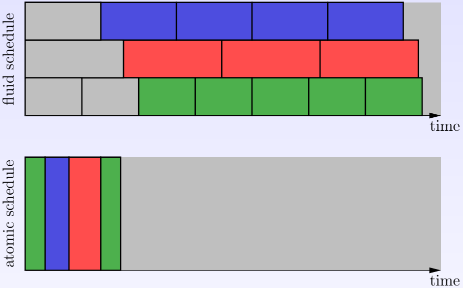
At each step, choose application which minimize

$$(n_k + 1) \times \frac{t_k}{\alpha_k}$$

n_k: number of task from application k already scheduled

One-dimensional load-balancing

- General fluid schedule with rate α_k for application k
- task of application k takes time t_k at full speed



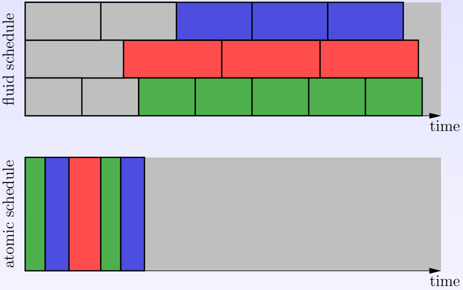
At each step, choose application which minimize

$$(n_k + 1) \times \frac{t_k}{\alpha_k}$$

n_k: number of task from application k already scheduled

One-dimensional load-balancing

- General fluid schedule with rate α_k for application k
- task of application k takes time t_k at full speed



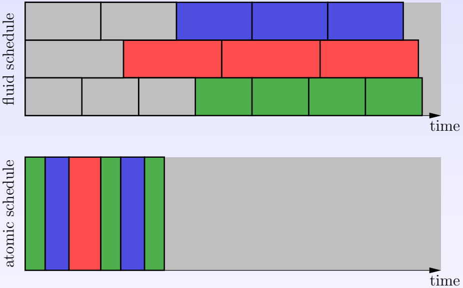
At each step, choose application which minimize

$$(n_k + 1) \times \frac{t_k}{\alpha_k}$$

n_k: number of task from application k already scheduled

One-dimensional load-balancing

- General fluid schedule with rate α_k for application k
- task of application k takes time t_k at full speed



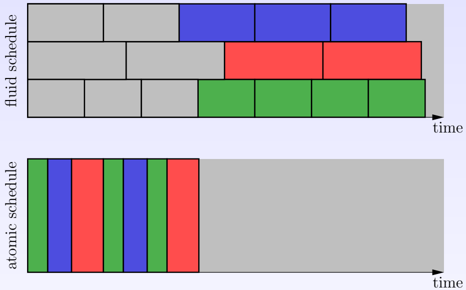
At each step, choose application which minimize

$$(n_k + 1) \times \frac{t_k}{\alpha_k}$$

n_k: number of task from application k already scheduled

One-dimensional load-balancing

- General fluid schedule with rate α_k for application k
- task of application k takes time t_k at full speed



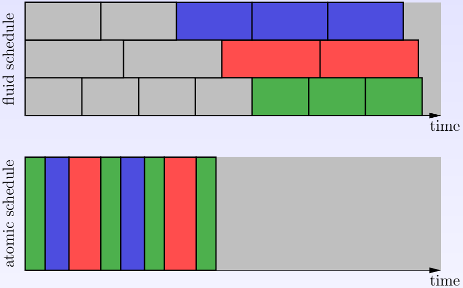
At each step, choose application which minimize

$$(n_k + 1) \times \frac{t_k}{\alpha_k}$$

n_k: number of task from application k already scheduled

One-dimensional load-balancing

- General fluid schedule with rate α_k for application k
- task of application k takes time t_k at full speed



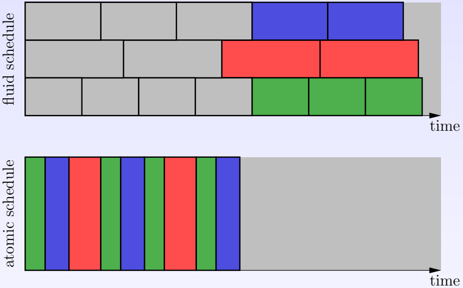
At each step, choose application which minimize

$$(n_k + 1) \times \frac{t_k}{\alpha_k}$$

n_k: number of task from application k already scheduled

One-dimensional load-balancing

- General fluid schedule with rate α_k for application k
- task of application k takes time t_k at full speed



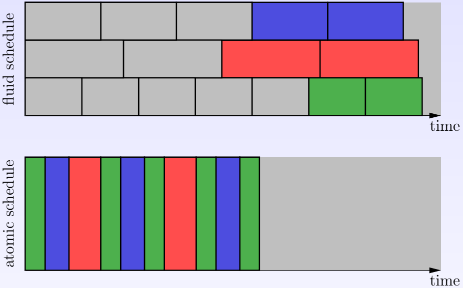
At each step, choose application which minimize

$$(n_k + 1) \times \frac{t_k}{\alpha_k}$$

n_k: number of task from application k already scheduled

One-dimensional load-balancing

- General fluid schedule with rate α_k for application k
- task of application k takes time t_k at full speed



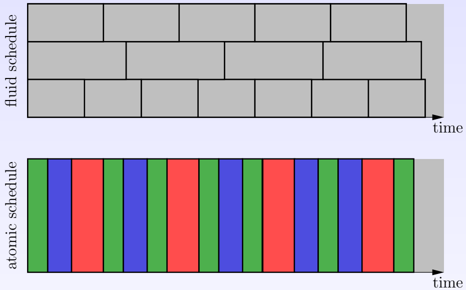
At each step, choose application which minimize

$$(n_k + 1) \times \frac{t_k}{\alpha_k}$$

n_k: number of task from application k already scheduled

One-dimensional load-balancing

- General fluid schedule with rate α_k for application k
- task of application k takes time t_k at full speed



At each step, choose application which minimize

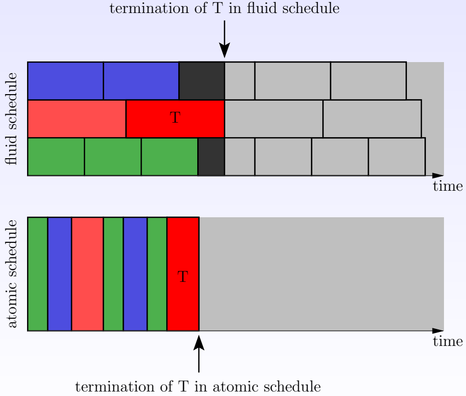
$$(n_k + 1) \times \frac{t_k}{\alpha_k}$$

n_k: number of task from application k already scheduled

Properties of 1D schedules

Lemma (1D).

In the 1D schedule, a task does not terminate later than in the fluid schedule.



Properties of 1D schedules

Lemma (1D).

In the 1D schedule, a task does not terminate later than in the fluid schedule.

Construction of 1D-inv schedule from a fluid schedule (M: Makespan):

1. Reverse the time: $t \rightsquigarrow M - t$
2. Apply 1D algorithm
3. Reverse the time once again

Lemma (1D-inv).

In the 1D-inv schedule, a task does not start earlier than in the fluid schedule, and 1D-inv has a makespan $\leq M$.

Back to the one-port model

From a fluid schedule (of communications and computations):

1. Round every quantities down to integer values
2. Shift all computations by one task (to cope with dependencies)
3. Apply 1D algorithm to communications
   $\longrightarrow$ communications finish in time
4. Apply 1D-inv algorithm to computations
   $\longrightarrow$ computations do not start in advance

Results:

- We guarantee that data dependencies are satisfied
- Some tasks may be forgotten

Back to the one-port model

Bound on the number of tasks not sent to P_u at time d_k:

- one per time-interval because of rounding

Time needed to send the remaining tasks:

$$\sum_{u=1}^{n} \frac{L_k \delta^{(k)}}{b_u}$$

Bound on the number of tasks not processed at P_u at time d_k:

- one per time-interval because of rounding (communications)
- one per time-interval because of rounding (computations)
- one because of shifting

Time needed to process these tasks:

$$\frac{(2L_k + 1) w^{(k)}}{s_u^{(k)}}$$

Back to the one-port model

Maximum delay for application A_k:

$$lateness^{(k)} \leq \sum_k \left(\sum_{u=1}^{n} \frac{L_k \delta^{(k)}}{b_u} + \max_{1 \leq u \leq n} \frac{(2L_k + 1)w^{(k)}}{s_u^{(k)}} \right)$$

Back to the one-port model

Maximum delay for application A_k:

$$\textit{lateness}^{(k)} \leq \sum_k \left(\sum_{u=1}^{n} \frac{L_k \delta^{(k)}}{b_u} + \max_{1 \leq u \leq n} \frac{(2L_k + 1) w^{(k)}}{s_u^{(k)}} \right)$$

Back to the one-port model: Asymptotic optimality

We introduce the granularity g:

$$\begin{aligned} \Pi_g^{(k)} &= \frac{\Pi^{(k)}}{g} \\ w_g^{(k)} &= g \times w^{(k)} \\ \delta_g^{(k)} &= g \times \delta^{(k)} \end{aligned}$$

- $g = 1$: no changes
- $g \to 0$: many small tasks (Divisible Load)

Theorem.

$$\textit{lateness}^{(k)} \xrightarrow[g \to 0]{} 0$$

When the granularity of the application gets smaller (many small tasks), the one-port makespan gets closer to the fluid makespan.

In practice:

- 1D schedule for communications
- Earliest Deadline First for computations

Back to the one-port model: Asymptotic optimality

We introduce the granularity g:

$$\begin{aligned}
\Pi_g^{(k)} &= \frac{\Pi^{(k)}}{g} \\
w_g^{(k)} &= g \times w^{(k)} \\
\delta_g^{(k)} &= g \times \delta^{(k)}
\end{aligned}$$

- $g = 1$: no changes
- $g \to 0$: many small tasks (Divisible Load)

Theorem.

$$\textit{lateness}^{(k)} \xrightarrow[g \to 0]{} 0$$

When the granularity of the application gets smaller (many small tasks), the one-port makespan gets closer to the fluid makespan.

In practice:

- 1D schedule for communications
- Earliest Deadline First for computations

Back to the one-port model: Asymptotic optimality

We introduce the granularity g:

$$\begin{aligned}\Pi_g^{(k)} &= \frac{\Pi^{(k)}}{g}\\ w_g^{(k)} &= g \times w^{(k)}\\ \delta_g^{(k)} &= g \times \delta^{(k)}\end{aligned}$$

- $g = 1$: no changes
- $g \to 0$: many small tasks (Divisible Load)

Theorem.

$$\textit{lateness}^{(k)} \xrightarrow[g \to 0]{} 0$$

When the granularity of the application gets smaller (many small tasks), the one-port makespan gets closer to the fluid makespan.

In practice:

- 1D schedule for communications
- Earliest Deadline First for computations

Outline

Online multi-application – framework

- No available information about future submission
- Information for application k available at release date $r^{(k)}$

Adaptation:

- Consider only available information (already submitted applications)
- Restart offline algorithm at each release date (with updated information)

- online heuristic named **CBS3M**-online
- we also test the offline algorithm: **CBS3M**-offline

Online multi-application – framework

Classical heuristics to prioritize applications:

- First In First Out (**FIFO**)
- Shortest Processing Time (**SPT**)
- Shortest Remaining Processing Time (**SRPT**)
- Shortest Weighted Remaining Processing Time (**SWRPT**)

($\times$ heuristic to chose workers: **RR**, **MCT** or **DD**)

Previous heuristics do not mix applications,

- Master-Worker Multi-Application (**MWMA**)
  (previous work, designed for simultaneous submissions)

Outline

Simulation and Experiment Settings

Experiments:

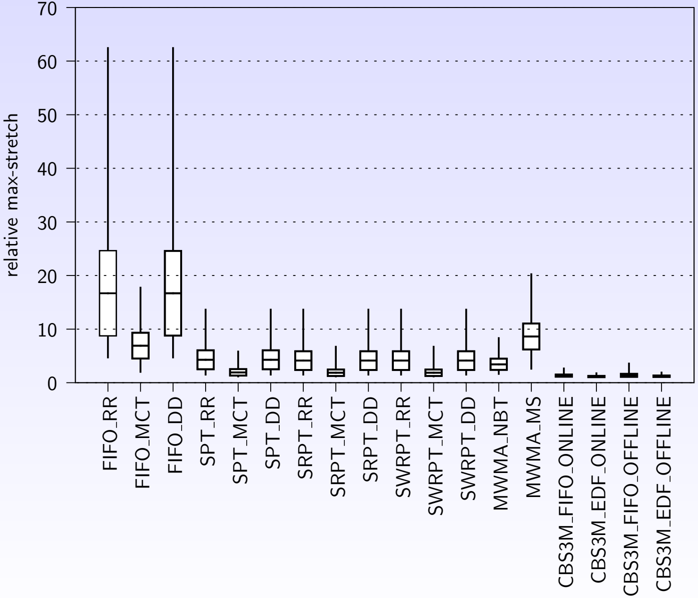
- GDSDMI cluster (8 workers)
- MPI communications
- Artificially slow-down communication and/or computations to emulate heterogeneity

Simulation:

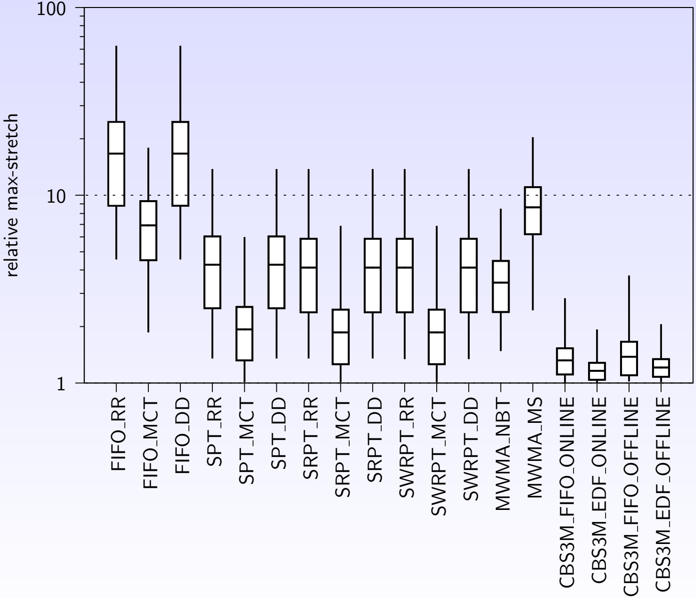
- SimGrid simulator
- Two scenarios:
  1. simulate MPI experiments
  2. extensive simulations with larger applications

parameters

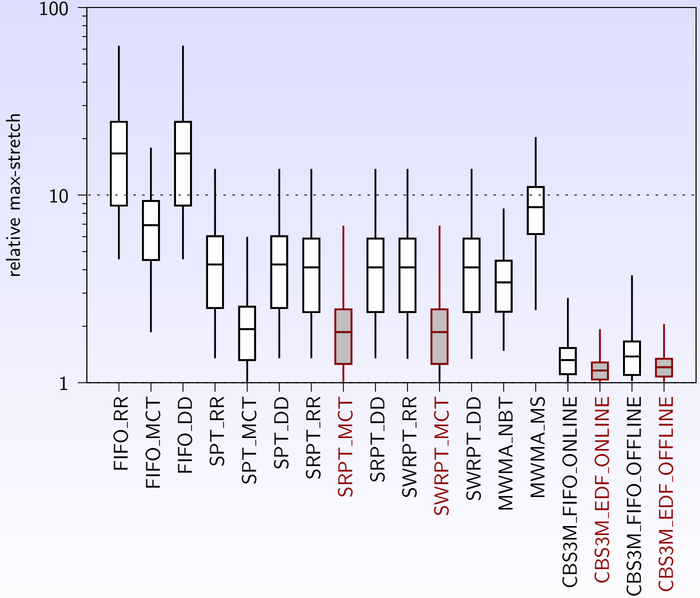
Simulations results



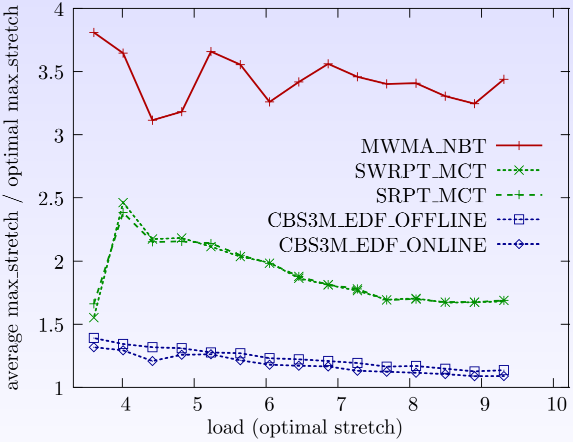
Simulations results



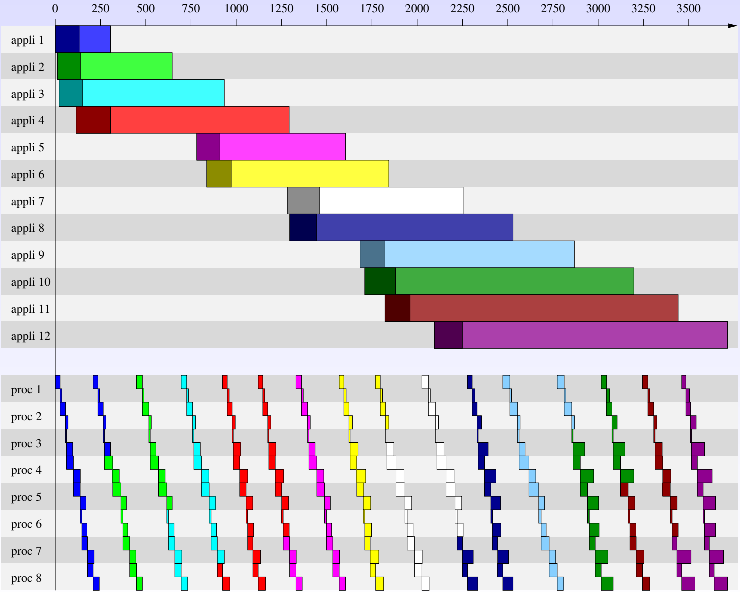
Simulations results



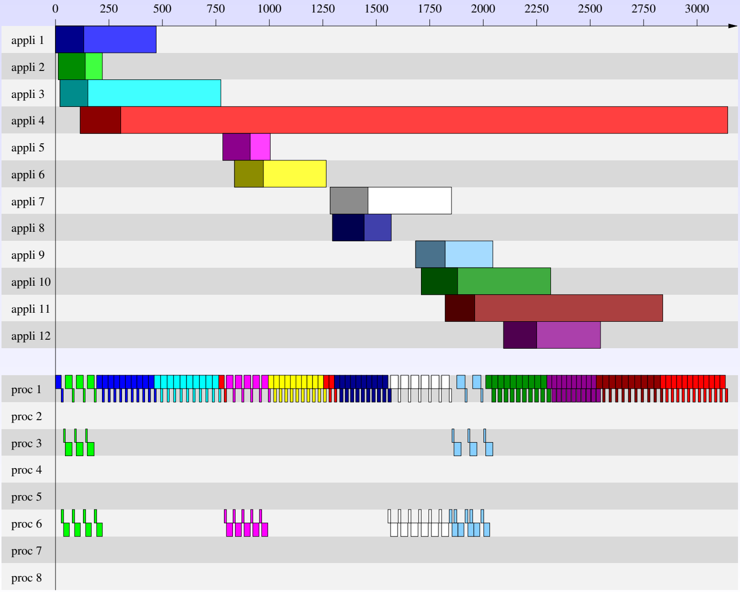
Simulations results – variation with load



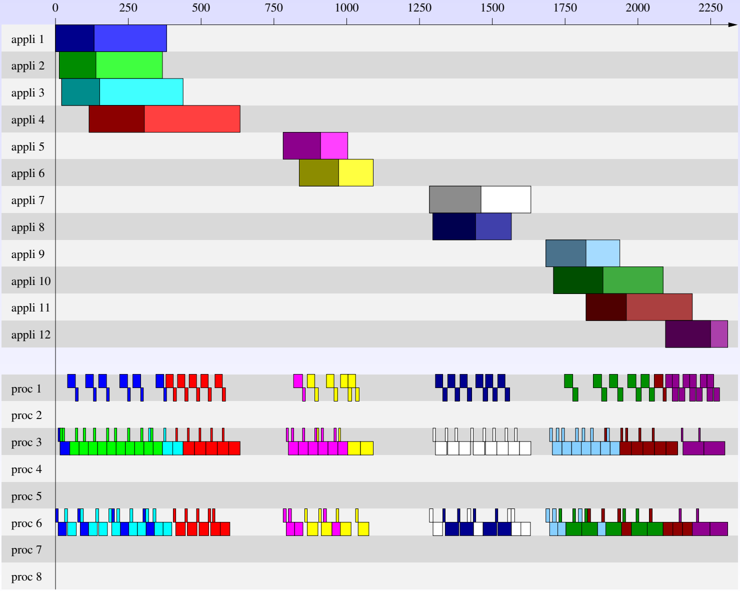
Gantt chart example: FIFO + RR



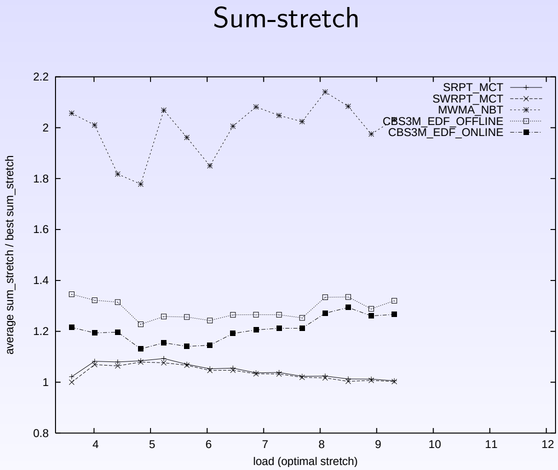
Gantt chart example: SRPT + MCT



Gantt chart example: CBS3M + EDF (online)



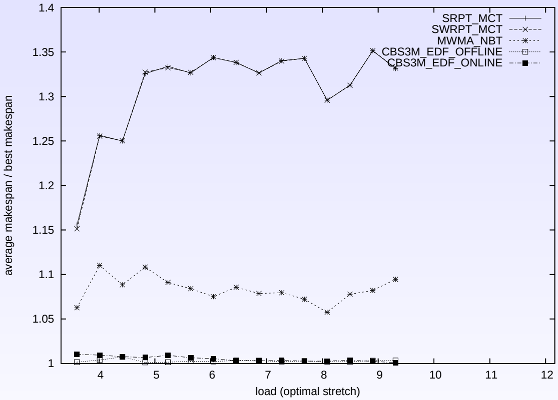
Simulations results – other metrics



- best strategy: **SWRPT** (known to be optimal)
- CBSSM within 30-40%

Simulations results – other metrics

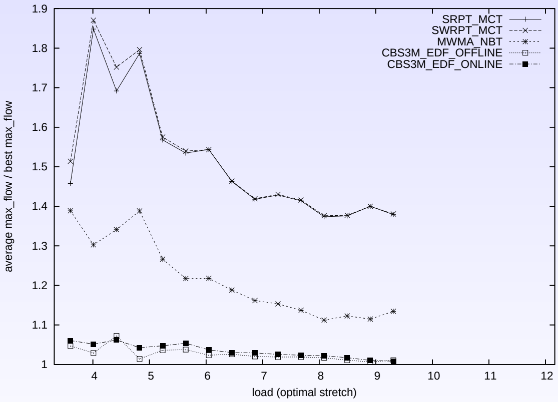
Makespan



- best strategy: **CBS3M**

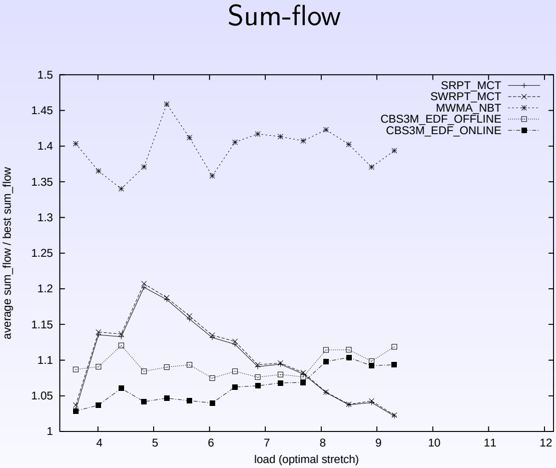
Simulations results – other metrics

Max-flow



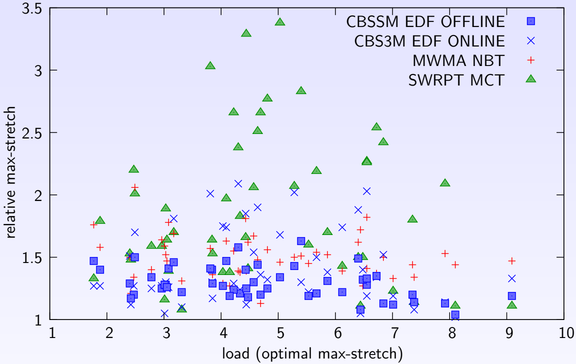
- best strategy: **CBS3M**

Simulations results – other metrics



- best strategy: **CBS3M**/ **SWRPT**

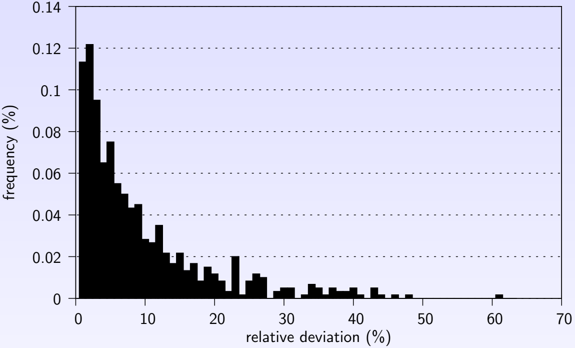
MPI experiments results



MPI experiments results

Algorithm	minimum	average	(± stddev)	maximum	(fraction of best result)
CBS3M_EDF_OFFLINE	**1.04**	**1.30**	**(± 0.13)**	**1.63**	**(the best in 38.0%)**
CBS3M_EDF_ONLINE	**1.02**	**1.41**	**(± 0.30)**	**2.09**	**(the best in 30.0%)**
CBS3M_FIFO_OFFLINE	1.04	1.38	(± 0.28)	2.97	(the best in 12.0%)
CBS3M_FIFO_ONLINE	1.02	1.46	(± 0.26)	1.96	(the best in 6.0%)
FIFO_MCT	1.10	1.81	(± 0.60)	4.15	(the best in 4.0%)
FIFO_RR	1.35	4.99	(± 3.46)	19.50	(the best in 0.0%)
MWMA_MS	1.22	2.29	(± 0.56)	4.05	(the best in 0.0%)
MWMA_NBT	1.13	1.50	(± 0.17)	2.06	(the best in 4.0%)
SPT_DD	1.33	4.87	(± 3.10)	18.75	(the best in 0.0%)
SPT_MCT	1.08	1.84	(± 0.61)	3.43	(the best in 4.0%)
SRPT_MCT	1.09	1.87	(± 0.59)	3.38	(the best in 0.0%)
SWRPT_MCT	1.08	1.88	(± 0.59)	3.38	(the best in 2.0%)

MPI experiments vs simulations



Relative deviation: $\dfrac{|\mathcal{S}_{\text{exp}} - \mathcal{S}_{\text{simu}}|}{\mathcal{S}_{\text{exp}}}$

- average difference: 8.9%
- standard deviation: 9.5%
- median value: 5.5%

Simulation and Experiment results – Summary

- **CBS3M** performs very well for max-stretch (best results in all cases, average ratio to the theoretical fluid optimal: 1.3, worst case: 2)
- **CBS3M** performs also well for other metrics: makespan, max-flow, (sum-stretch, sum-flow)
- Explanation: makes good use of the platform (like MWMA) and is aware of the priorities/deadlines (like SWRPT)
- Simulation/Experiments close enough: average relative deviation 8.9%, median 5.5%

Outline

Conclusion

- Key points:
  - Realistic platform model
  - Optimal offline algorithm
    (asymptotically optimal in the one-port model)
  - Efficient online algorithm based on offline study

- Extensions:
  - Extend the simulation to larger platform
  - Bi-criteria
  - Robustness

Positive values

- **Non-negative throughputs.**

$$\forall 1 \leq u \leq p, \forall 1 \leq k \leq n, \forall 1 \leq j \leq 2n-1,$$
$$\rho^{(k)}_{M \to u}(t_j, t_{j+1}) \geq 0 \text{ and } \rho^{(k)}_{u}(t_j, t_{j+1}) \geq 0. \quad (1)$$

- **Non-negative buffers.**

$$\forall\ 1 \leq k \leq n, \forall 1 \leq u \leq p, \forall 1 \leq j \leq 2n,$$
$$B^{(k)}_{u}(t_j) \geq 0. \quad (2)$$

Physical constraints

- **Bounded link capacity.**

$$\forall 1 \leq j \leq 2n-1, \forall 1 \leq u \leq p,$$

$$\sum_{k=1}^{n} \rho_{M\to u}^{(k)}(t_j, t_{j+1}) \frac{\delta^{(k)}}{b_u} \leq 1. \quad (3)$$

- **Limited sending capacity of master.**

$$\forall 1 \leq j \leq 2n-1,$$

$$\sum_{u=1}^{p} \sum_{k=1}^{n} \rho_{M\to u}^{(k)}(t_j, t_{j+1}) \frac{\delta^{(k)}}{\mathcal{B}} \leq 1. \quad (4)$$

- **Bounded computing capacity.**

$$\forall 1 \leq j \leq 2n-1, \forall 1 \leq u \leq p,$$

$$\sum_{k=1}^{n} \rho_{u}^{(k)}(t_j, t_{j+1}) \frac{w^{(k)}}{s_u^{(k)}} \leq 1. \quad (5)$$

Buffer constraints

- **Buffer initialization.**

$$\forall\ 1 \leq k \leq n, \forall 1 \leq u \leq p,$$
$$B_u^{(k)}(r^{(k)}) = 0. \quad (6)$$

- **Emptying Buffer.**

$$\forall\ 1 \leq k \leq n, \forall 1 \leq u \leq p,$$
$$B_u^{(k)}(d^{(k)}) = 0. \quad (7)$$

- **Bounded size**

$$\forall 1 \leq u \leq p, \forall 1 \leq j \leq 2n,$$
$$\sum_{k=1}^{n} B_u^{(k)}(t_j)\delta^{(k)} \leq M_u. \quad (8)$$

Tasks constraints

- **Task conservation.**

$$\forall\ 1 \le k \le n, \forall 1 \le j \le 2n-1, \forall 1 \le u \le p,$$
$$B_u^{(k)}(t_{j+1}) = B_u^{(k)}(t_j) + \big(\rho_{M\to u}^{(k)}(t_j, t_{j+1}) - \rho_u^{(k)}(t_j, t_{j+1})\big) \times \big(t_{j+1} - t_j\big). \tag{9}$$

- **Total number of tasks.**

$$\forall\ 1 \le k \le n,$$
$$\sum_{\substack{1 \le j \le 2n-1 \\ t_j \ \ge\ r^{(k)} \\ t_{j+1}\ \le\ d^{(k)}}} \sum_{u=1}^{p} \rho_{M\to u}^{(k)}(t_j, t_{j+1}) \times (t_{j+1} - t_j) = \Pi^{(k)}. \tag{10}$$

Polyhedron

$$\begin{cases} \text{find } \rho^{(k)}_{M\to u}(t_j, t_{j+1}), \rho^{(k)}_{u}(t_j, t_{j+1}), \\ \forall k, u, j \text{ such that } 1 \le k \le n, 1 \le u \le p, 1 \le j \le 2n-1 \\ \text{under the constraints } (1), (2), (3), (4), (5), (6), (7), (8), (9) \text{ and } (10) \end{cases} \quad (K)$$

A given max-stretch $\mathcal{S}'$ is achievable if and only if the Polyhedron (K) is not empty

In practice, we add a fictitious linear objective function. Back

New constraints

- **Bounded link capacity.**

$$\forall 1 \leq j \leq 2n-1, \forall 1 \leq u \leq p,$$
$$\sum_{k=1}^{n} A^{(k)}_{M \to u}(t_j, t_{j+1}) \frac{\delta^{(k)}}{b_u} \leq (\alpha_{j+1} - \alpha_j)\mathcal{S} + (\beta_{j+1} - \beta_j)$$

◂ Back

New constraints

- **Bounded link capacity.**
- **Limited sending capacity of master.**

$$\forall 1 \leq j \leq 2n-1,$$
$$\sum_{u=1}^{p} \sum_{k=1}^{n} A_{M \to u}^{(k)}(t_j, t_{j+1}) \delta^{(k)} \leq \mathcal{B} \times \left((\alpha_{j+1} - \alpha_j) \mathcal{S} + (\beta_{j+1} - \beta_j) \right)$$

Back

New constraints

- **Bounded link capacity.**
- **Limited sending capacity of master.**
- **Bounded computing capacity.**

$$\forall 1 \leq j \leq 2n-1, \forall 1 \leq u \leq p,$$
$$\sum_{k=1}^{n} A_u^{(k)}(t_j, t_{j+1}) \frac{w^{(k)}}{s_u^{(k)}} \leq (\alpha_{j+1} - \alpha_j)\mathcal{S} + (\beta_{j+1} - \beta_j)$$

Back

New constraints

- **Bounded link capacity.**
- **Limited sending capacity of master.**
- **Bounded computing capacity.**
- **Total number of tasks.**

$$\forall\ 1 \leq k \leq n,$$

$$\sum_{\substack{1 \leq j \leq 2n-1 \\ t_j \ \geq \ r^{(k)} \\ t_{j+1} \ \leq \ d^{(k)}}} \sum_{u=1}^{p} A^{(k)}_{M \to u}(t_j, t_{j+1}) = \Pi^{(k)}$$

Back

New constraints

- **Bounded link capacity.**
- **Limited sending capacity of master.**
- **Bounded computing capacity.**
- **Total number of tasks.**
- **Task conservation.**

$$\forall\ 1 \leq k \leq n, \forall 1 \leq j \leq 2n-1, \forall 1 \leq u \leq p,$$
$$B_u^{(k)}(t_{j+1}) = B_u^{(k)}(t_j) + A_{M \to u}^{(k)}(t_j, t_{j+1}) - A_u^{(k)}(t_j, t_{j+1})$$

Back

New constraints

- Bounded link capacity.
- Limited sending capacity of master.
- Bounded computing capacity.
- Total number of tasks.
- Task conservation.
- Non-negative buffer.
- Buffer initialization.
- Emptying Buffer.

Back

New constraints

- Bounded link capacity.
- Limited sending capacity of master.
- Bounded computing capacity.
- Total number of tasks.
- Task conservation.
- Non-negative buffer.
- Buffer initialization.
- Emptying Buffer.
- Bounded stretch

$$\mathcal{S}_a \leq \mathcal{S} \leq \mathcal{S}_b \tag{11}$$

Back

Parameters for the MPI experiments and for the SimG

	parameter	experiments	simulations
general	number of workers	8	10
	number of applications	12	20
arrival dates	mean of the distribution in the log space	4.0	4.0
	standard deviation in the log space	1.2	1.2
computations	maximum amount of work application (Gflops)	76.8	409
	minimum amount of work per task (Gflops)	3.1	3.1
communications	maximum amount of communication per application (MB)	800	6,000
	minimum amount of communication per task (MB)	40	40
number of tasks	minimum number of tasks per application	10	20

Back