Ordonnancement sur plates-formes hétérogènes de tâches partageant des données

Thèse de doctorat

Arnaud Giersch

22 décembre 2004

Introduction

- D'importants besoins en puissance de calcul.
- Le parallélisme est une réponse possible.
- L'hétérogénéité des plates-formes actuelles rend les problèmes d'ordonnancement encore plus difficiles.
- Les données sont potentiellement partagées par des calculs différents.

Introduction

- D'importants besoins en puissance de calcul.
- Le parallélisme est une réponse possible.

- L'hétérogénéité des plates-formes actuelles rend les problèmes d'ordonnancement encore plus difficiles.
- Les données sont potentiellement partagées par des calculs différents.

Introduction

- D'importants besoins en puissance de calcul.
- Le parallélisme est une réponse possible.

- L'hétérogénéité des plates-formes actuelles rend les problèmes d'ordonnancement encore plus difficiles.
- Les données sont potentiellement partagées par des calculs différents.

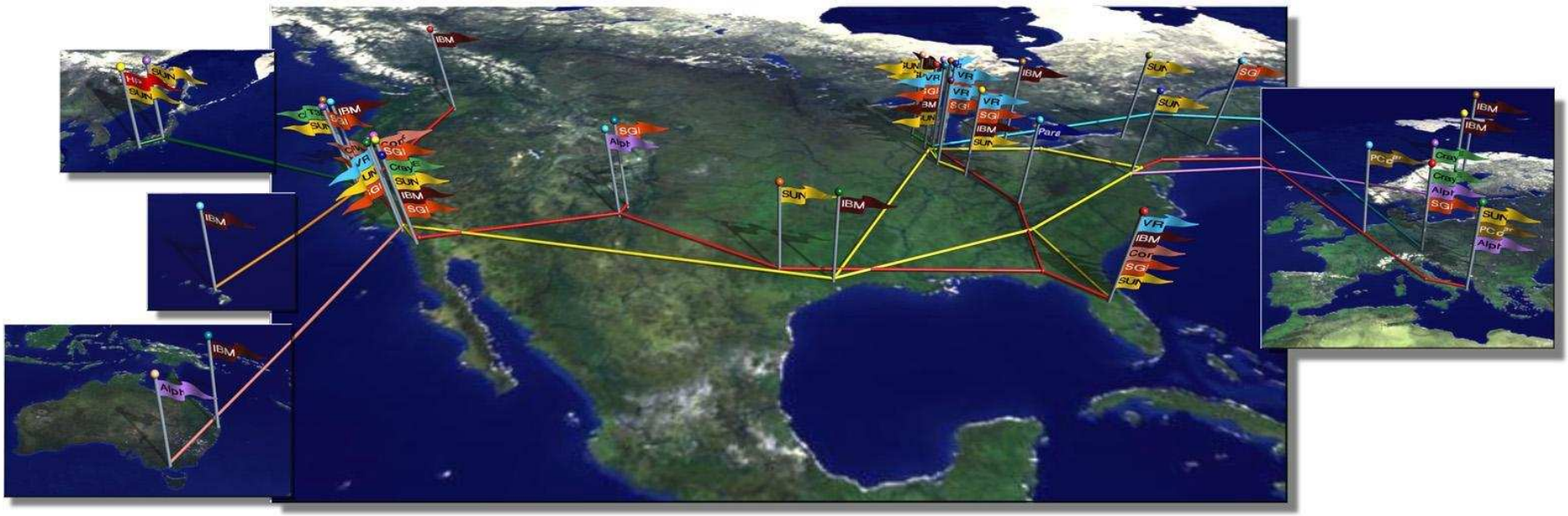
Introduction

- D'importants besoins en puissance de calcul.
- Le parallélisme est une réponse possible.

- L'hétérogénéité des plates-formes actuelles rend les problèmes d'ordonnancement encore plus difficiles.
- Les données sont potentiellement partagées par des calculs différents.

Introduction

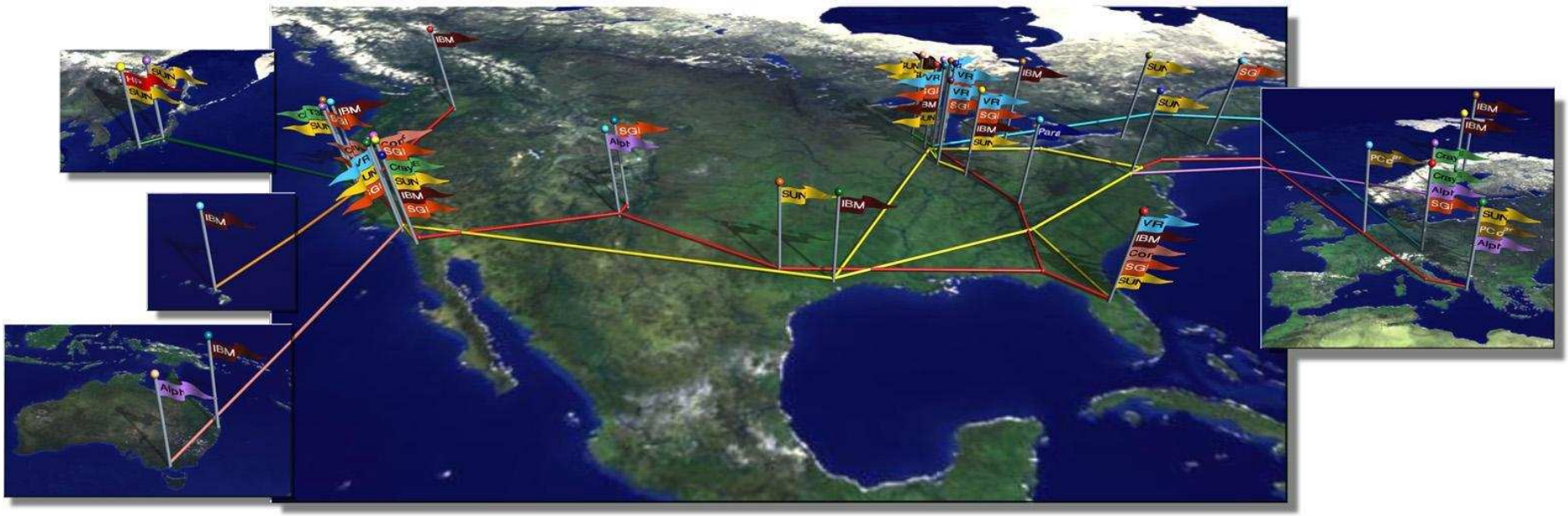
- D'importants besoins en puissance de calcul.
- Le parallélisme est une réponse possible.



- L'hétérogénéité des plates-formes actuelles rend les problèmes d'ordonnancement encore plus difficiles.
- Les données sont potentiellement partagées par des calculs différents.

Introduction

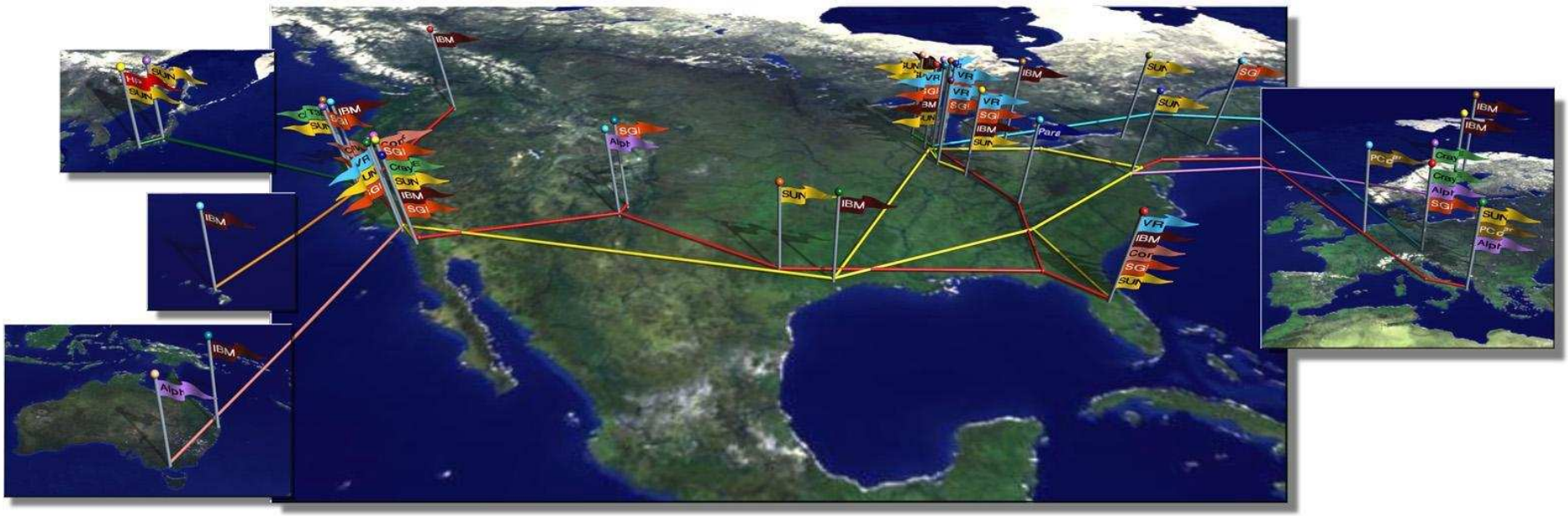
- D'importants besoins en puissance de calcul.
- Le parallélisme est une réponse possible.



- L'hétérogénéité des plates-formes actuelles rend les problèmes d'ordonnancement encore plus difficiles.
- Les données sont potentiellement partagées par des calculs différents.

Introduction

- D'importants besoins en puissance de calcul.
- Le parallélisme est une réponse possible.



- L'hétérogénéité des plates-formes actuelles rend les problèmes d'ordonnancement encore plus difficiles.
- Les données sont potentiellement partagées par des calculs différents.

Plan du manuscrit

Plan du manuscrit

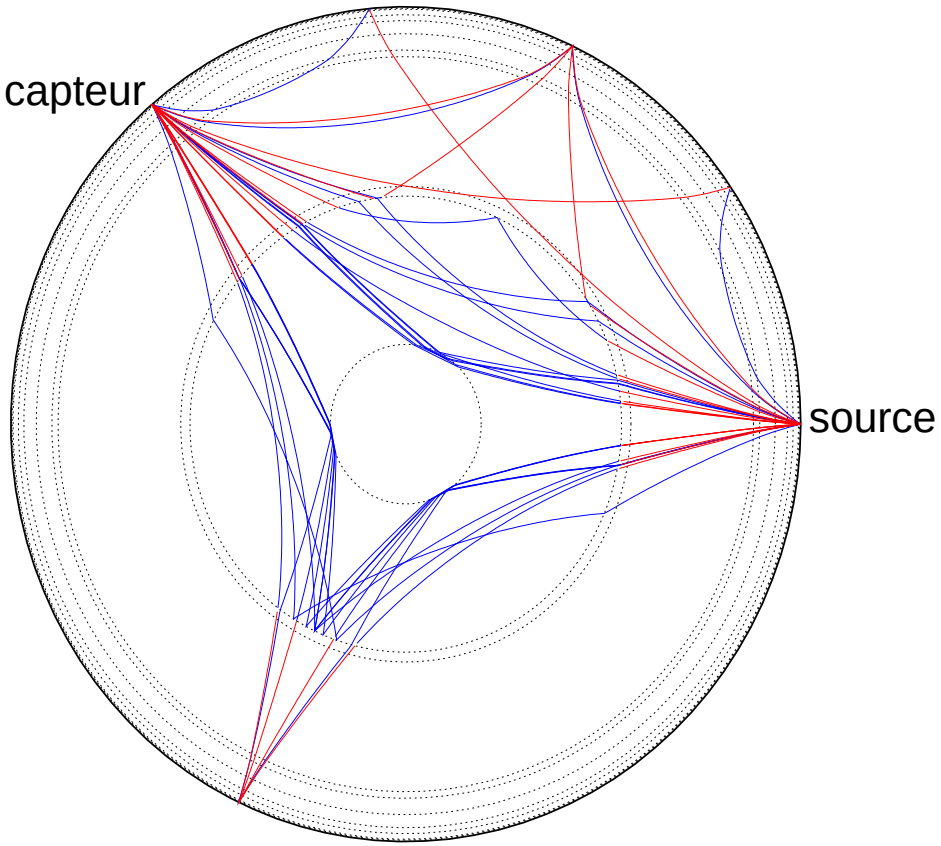
Maître-esclave

Application cible
Description (1/2)

- Application en tomographie sismique.
- But : modéliser la structure interne de la Terre du point de vue des vitesses de propagation des ondes sismiques.
- Utilise des informations recueillies lors d'évènements sismiques.
- Simulation de la propagation des ondes sismiques à travers un modèle de la Terre.

Application cible
Description (2/2)

- Tracé du chemin d'une onde (*rai*) à partir des coordonnées de la source et de la destination, et du type de l'onde.
- Le modèle de vitesses est corrigé en fonction des différences entre simulation et réalité.

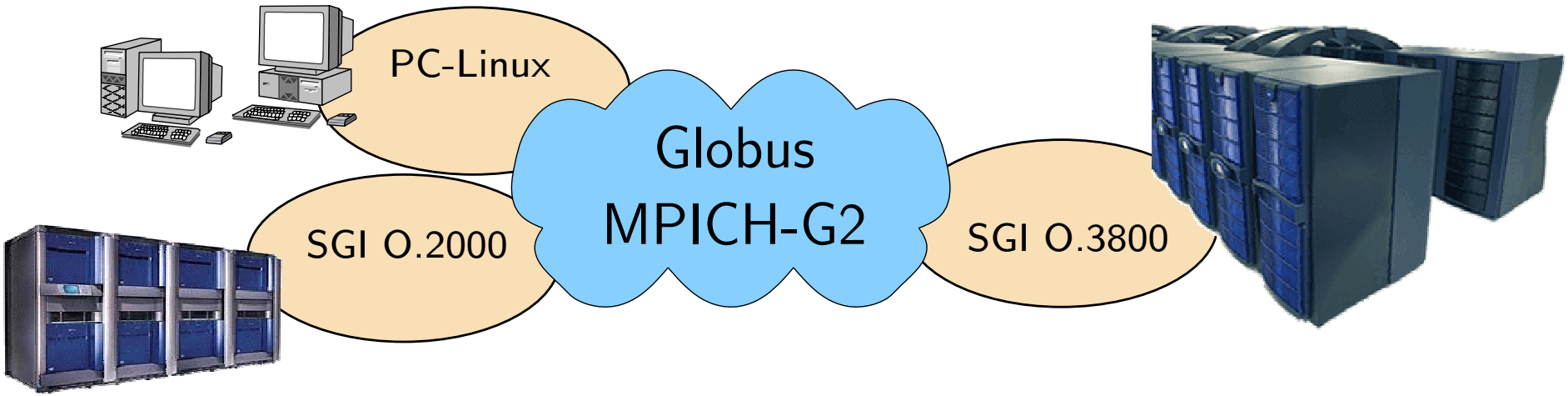


Application cible
Implémentation existante

- Données en entrée : caractéristiques des ondes sismiques.
- Architecture cible : machine parallèle homogène.
- Le tracé des différents rais peut être fait de manière indépendante.
- Application de type maître-esclave.

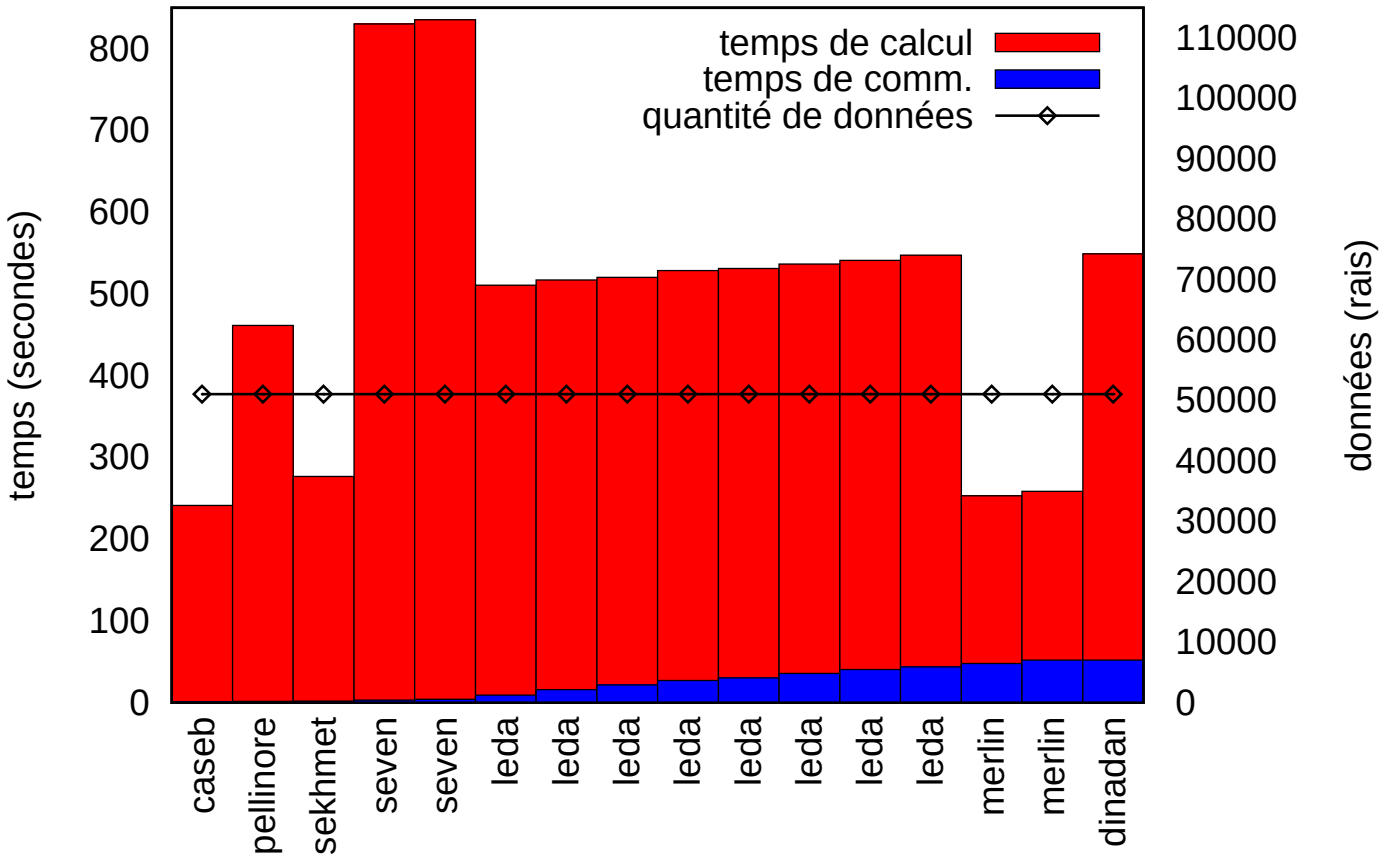
```
if (rank = ROOT)
    raydata ← lire d lignes du fichier de données ;
MPI_Scatter(raydata, d/P, ..., rbuff, ...,
            ROOT, MPI_COMM_WORLD) ;
calculer(rbuff) ;
```

Environnement



Machine	Localisation	Type	CPU (ms/rai)	NET (μs/rai)
dinadan (maître)	Strasbourg	1 × PIII/933	9,29	0,0
caseb	Strasbourg	1 × XP1800	4,63	10,0
pellinore	Strasbourg	1 × PIII/800	9,37	11,2
sekhmet	Strasbourg	1 × XP1800	4,89	17,0
seven	Strasbourg	2 × R12K/300	16,16	21,0
leda	Montpellier	8 × R14K/500	9,68	35,3
merlin	Strasbourg	2 × XP2000	3,98	81,5

Répartition uniforme des données

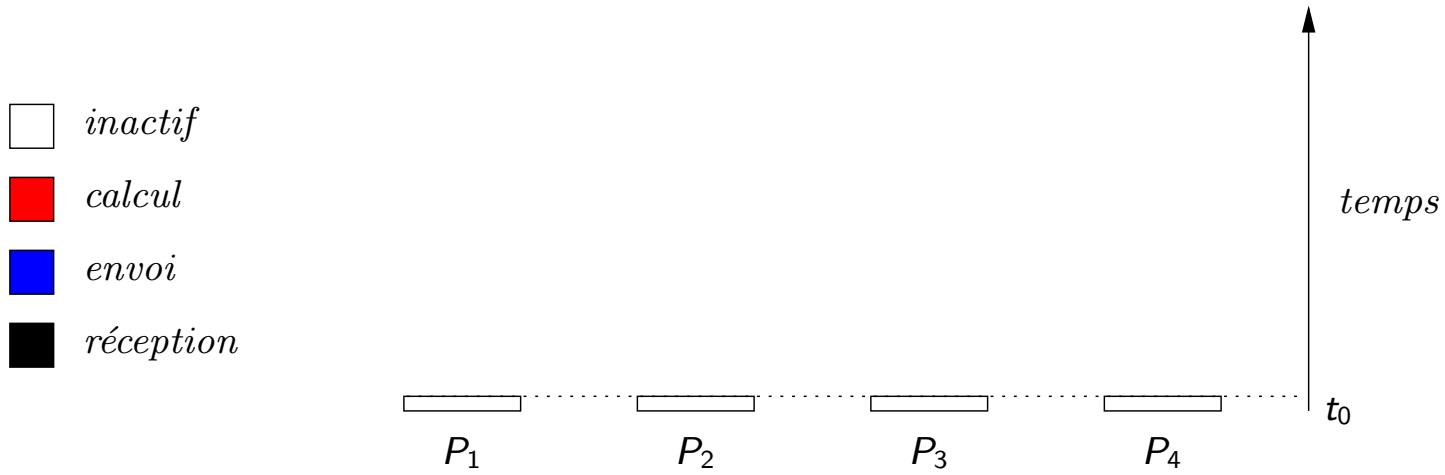


Question

Comment équilibrer l'exécution ?

Opération de distribution

- Une opération de distribution suivie d'une phase de calcul



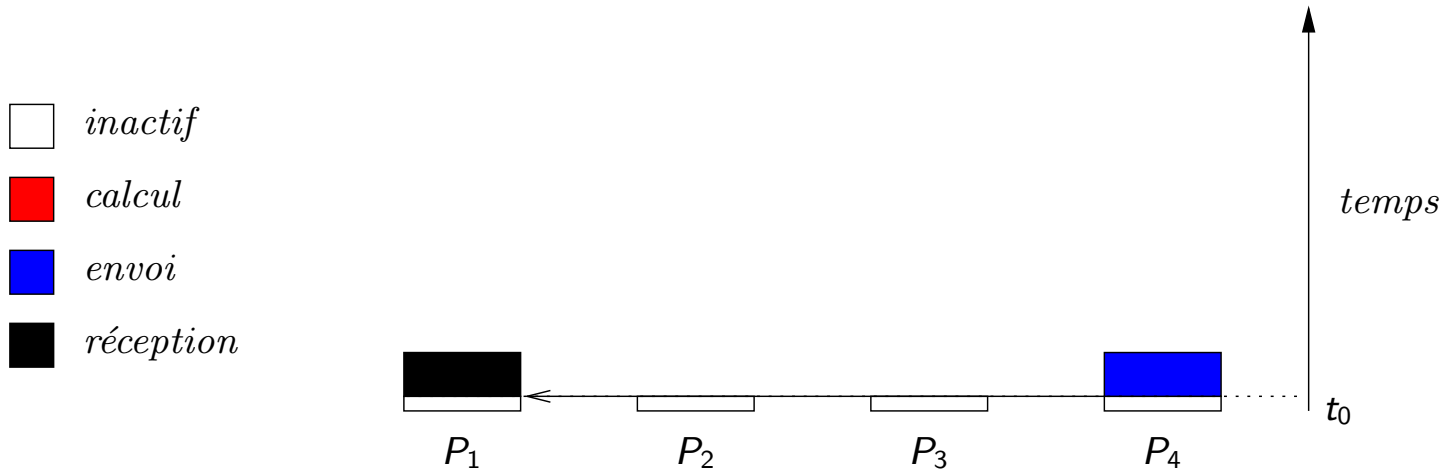
- Maître : P_4 (modèle un-port)

Questions

- comment répartir les données sur les processeurs ?
- dans quel ordre envoyer les données aux processeurs ?

Opération de distribution

- Une opération de distribution suivie d'une phase de calcul



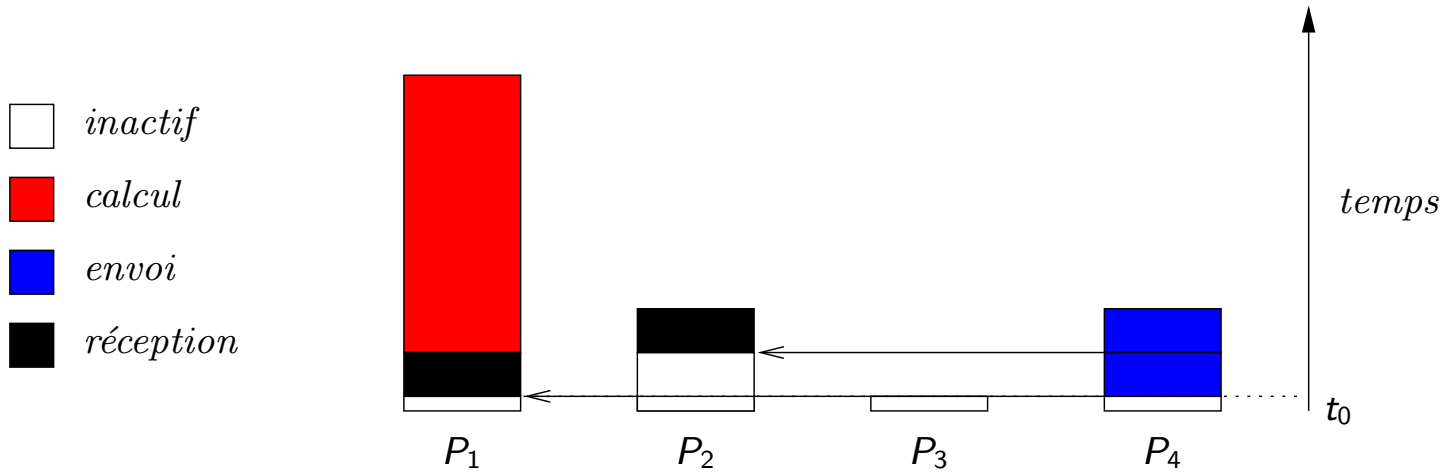
- Maître : P_4 (modèle un-port)

Questions

- comment répartir les données sur les processeurs ?
- dans quel ordre envoyer les données aux processeurs ?

Opération de distribution

- Une opération de distribution suivie d'une phase de calcul



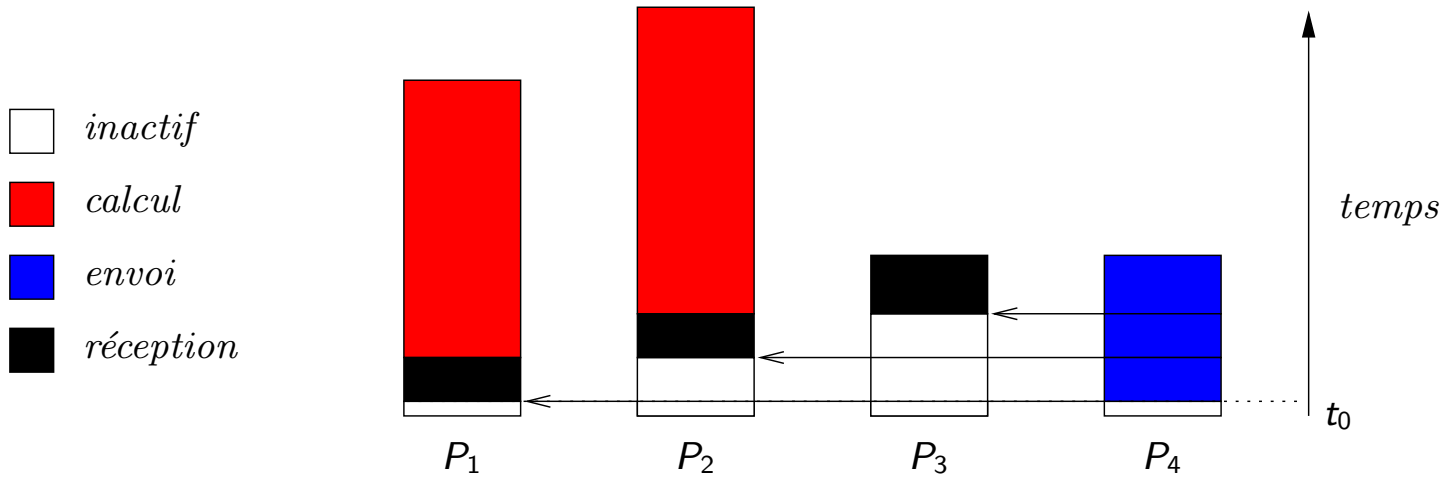
- Maître : P_4 (modèle un-port)

Questions

- comment répartir les données sur les processeurs ?
- dans quel ordre envoyer les données aux processeurs ?

Opération de distribution

- Une opération de distribution suivie d'une phase de calcul



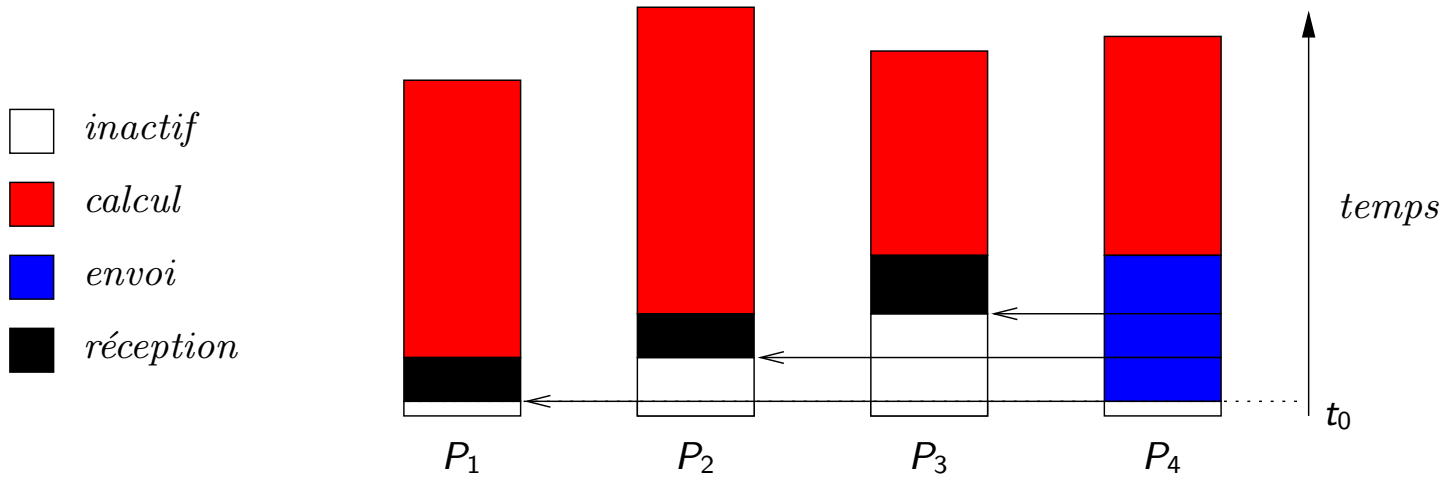
- Maître : P_4 (modèle un-port)

Questions

- comment répartir les données sur les processeurs ?
- dans quel ordre envoyer les données aux processeurs ?

Opération de distribution

- Une opération de distribution suivie d'une phase de calcul



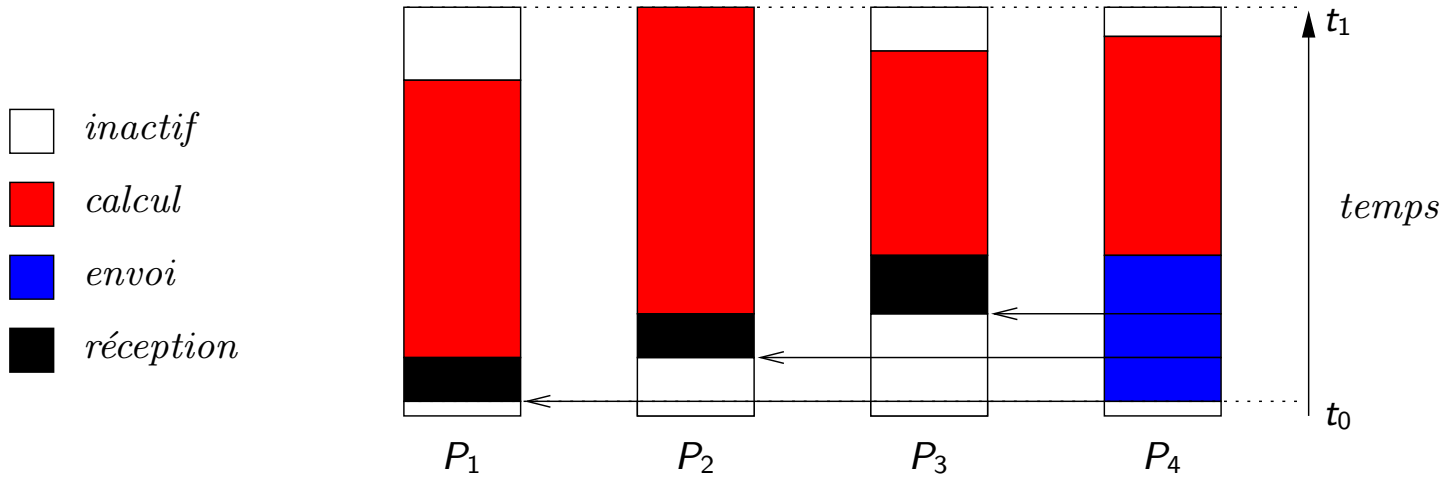
- Maître : P_4 (modèle un-port)

Questions

- comment répartir les données sur les processeurs ?
- dans quel ordre envoyer les données aux processeurs ?

Opération de distribution

- Une opération de distribution suivie d'une phase de calcul



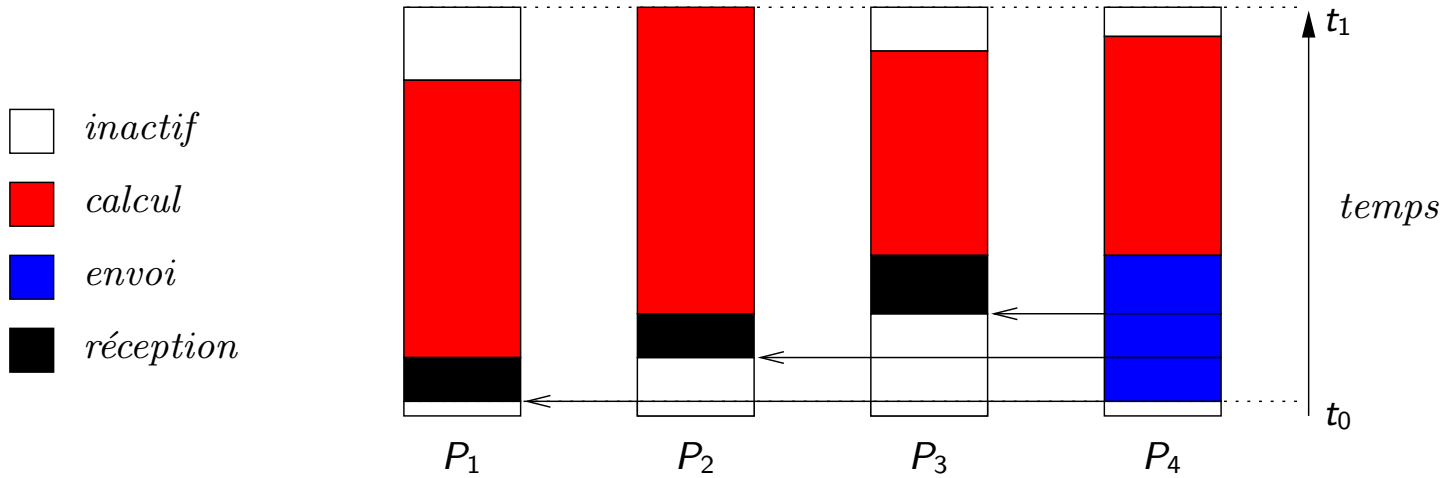
- Maître : P_4 (modèle un-port)

Questions

- comment répartir les données sur les processeurs ?
- dans quel ordre envoyer les données aux processeurs ?

Opération de distribution

- Une opération de distribution suivie d'une phase de calcul



- Maître : P_4 (modèle un-port)

Questions

- comment répartir les données sur les processeurs ?
- dans quel ordre envoyer les données aux processeurs ?

Modèles

- Processeurs : $P_1, \ldots, P_p$; distribution des données : $d_1, \ldots, d_p$
- Fonctions de coût : $T_{comm}(i, d)$ et $T_{calc}(i, d)$
- P_i termine son calcul à la date :

$$T_i = \sum_{j=1}^{i} T_{comm}(j, d_j) + T_{calc}(i, d_i)$$

- Temps total d'exécution :

$$T = \max_{1 \leq i \leq p} \left(\sum_{j=1}^{i} T_{comm}(j, d_j) + T_{calc}(i, d_i) \right)$$

- Distribution des donnée $d_1, \ldots, d_p$ minimisant T ?

Modèles

- Processeurs : $P_1, \ldots, P_p$; distribution des données : $d_1, \ldots, d_p$
- Fonctions de coût : $T_{comm}(i, d)$ et $T_{calc}(i, d)$
- P_i termine son calcul à la date :

$$T_i = \sum_{j=1}^{i} T_{comm}(j, d_j) + T_{calc}(i, d_i)$$

- Temps total d'exécution :

$$T = \max_{1 \leq i \leq p} \left(\sum_{j=1}^{i} T_{comm}(j, d_j) + T_{calc}(i, d_i) \right)$$

- Distribution des donnée $d_1, \ldots, d_p$ minimisant T ?

Modèles

- Processeurs : $P_1, \ldots, P_p$; distribution des données : $d_1, \ldots, d_p$
- Fonctions de coût : $T_{comm}(i, d)$ et $T_{calc}(i, d)$
- P_i termine son calcul à la date :

$$T_i = \sum_{j=1}^{i} T_{comm}(j, d_j) + T_{calc}(i, d_i)$$

- Temps total d'exécution :

$$T = \max_{1 \leq i \leq p} \left(\sum_{j=1}^{i} T_{comm}(j, d_j) + T_{calc}(i, d_i) \right)$$

- Distribution des donnée $d_1, \ldots, d_p$ minimisant T ?

Modèles

- Processeurs : $P_1, \ldots, P_p$; distribution des données : $d_1, \ldots, d_p$
- Fonctions de coût : $T_{comm}(i, d)$ et $T_{calc}(i, d)$
- P_i termine son calcul à la date :

$$T_i = \sum_{j=1}^{i} T_{comm}(j, d_j) + T_{calc}(i, d_i)$$

- Temps total d'exécution :

$$T = \max_{1 \leq i \leq p} \left(\sum_{j=1}^{i} T_{comm}(j, d_j) + T_{calc}(i, d_i) \right)$$

- Distribution des donnée $d_1, \ldots, d_p$ minimisant T ?

Modèles

- Processeurs : $P_1, \ldots, P_p$; distribution des données : $d_1, \ldots, d_p$
- Fonctions de coût : $T_{comm}(i, d)$ et $T_{calc}(i, d)$
- P_i termine son calcul à la date :

$$T_i = \sum_{j=1}^{i} T_{comm}(j, d_j) + T_{calc}(i, d_i)$$

- Temps total d'exécution :

$$T = \max_{1 \leq i \leq p} \left(\sum_{j=1}^{i} T_{comm}(j, d_j) + T_{calc}(i, d_i) \right)$$

- Distribution des donnée $d_1, \ldots, d_p$ minimisant T ?

Résolution exacte

Relation de récurrence

$$T_{opt}(d, P_i, \ldots, P_p) = \min_{0 \leq d_i \leq d} \left(T_{comm}(i, d_i) + \max \begin{pmatrix} T_{calc}(i, d_i) \\ T_{opt}(d - d_i, P_{i+1}, \ldots, P_p) \end{pmatrix} \right)$$

Algorithme par programmation dynamique

Pour $i \leftarrow p - 1$ **à** 1 **par pas de** -1

- connaissant une répartition optimale de 0 à d données sur $P_{i+1}, \ldots, P_p$
- calculer une répartition optimale de 0 à d données sur $P_i, \ldots, P_p$ en faisant varier le nombre de données attribuées à P_i

- Complexité algorithmique : $O(d^2 \cdot p)$
- Hypothèses :
  $T_{comm}(i, d)$ et $T_{calc}(i, d)$ sont positives, et nulles quand $d = 0$

Résolution exacte

Relation de récurrence

$$T_{opt}(d, P_i, \ldots, P_p) = \min_{0 \leq d_i \leq d} \left(T_{comm}(i, d_i) + \max \begin{pmatrix} T_{calc}(i, d_i) \\ T_{opt}(d - d_i, P_{i+1}, \ldots, P_p) \end{pmatrix} \right)$$

Algorithme par programmation dynamique

Pour $i \leftarrow p - 1$ **à** 1 **par pas de** -1

- connaissant une répartition optimale de 0 à d données sur $P_{i+1}, \ldots, P_p$
- calculer une répartition optimale de 0 à d données sur $P_i, \ldots, P_p$ en faisant varier le nombre de données attribuées à P_i

- Complexité algorithmique : $O(d^2 \cdot p)$
- Hypothèses :
  $T_{comm}(i, d)$ et $T_{calc}(i, d)$ sont positives, et nulles quand $d = 0$

Résolution exacte

Relation de récurrence

$$T_{opt}(d, P_i, \ldots, P_p) = \min_{0 \leq d_i \leq d} \left(T_{comm}(i, d_i) + \max \begin{pmatrix} T_{calc}(i, d_i) \\ T_{opt}(d - d_i, P_{i+1}, \ldots, P_p) \end{pmatrix} \right)$$

Algorithme par programmation dynamique

Pour $i \leftarrow p - 1$ **à** 1 **par pas de** -1

- connaissant une répartition optimale de 0 à d données sur $P_{i+1}, \ldots, P_p$
- calculer une répartition optimale de 0 à d données sur $P_i, \ldots, P_p$ en faisant varier le nombre de données attribuées à P_i

- Complexité algorithmique : $O(d^2 \cdot p)$
- Hypothèses :
  $T_{comm}(i, d)$ et $T_{calc}(i, d)$ sont positives, et nulles quand $d = 0$

Approches plus rapides

- L'algorithme peut être très long avec beaucoup de données (dans notre application : 817 101 rais).
- Des hypothèse supplémentaires sur les fonctions de coût permettent des approches plus efficaces.

Méthode	Fonctions de coût	Temps d'exécution
Algorithme 2.1	positives et nulles en $d = 0$	> 2 jours
Algorithme 2.2	positives, croissantes et nulles en $d = 0$	~ 6 minutes
Heuristique garantie	affines	instantané

- Ordre des processeurs ?

Approches plus rapides

- L'algorithme peut être très long avec beaucoup de données (dans notre application : 817 101 rais).
- Des hypothèse supplémentaires sur les fonctions de coût permettent des approches plus efficaces.

Méthode	Fonctions de coût	Temps d'exécution
Algorithme 2.1	positives et nulles en $d = 0$	$>$ 2 jours
Algorithme 2.2	positives, croissantes et nulles en $d = 0$	$\sim$ 6 minutes
Heuristique garantie	affines	instantané

- Ordre des processeurs ?

Tâches divisibles

- $T_{comm}(i,d) = \lambda_i \cdot d$ et $T_{calc}(i,d) = \mu_i \cdot d$
- Recherche d'une solution rationnelle.
  - Soit
    $$D(P_1,\ldots,P_p) = 1/\left(\sum_{i=1}^{p} \frac{1}{\lambda_i+\mu_i} \cdot \prod_{j=1}^{i-1} \frac{\mu_j}{\lambda_j+\mu_j}\right).$$
  - Il existe une solution rationnelle optimale où tous les processeurs travaillent si et seulement si
    $$\forall i \in [1, p-1], \quad \lambda_i \leq D(P_{i+1},\ldots,P_p).$$
  - Dans ce cas, dans une solution optimale, tous les processeurs terminent en même temps et :
    $$T = d \cdot D(P_1,\ldots,P_p)\,; \qquad d_i = \frac{1}{\lambda_i+\mu_i} \cdot \left(\prod_{j=1}^{i-1} \frac{\mu_j}{\lambda_j+\mu_j}\right) \cdot T.$$

Tâches divisibles

- $T_{comm}(i,d) = \lambda_i \cdot d$ et $T_{calc}(i,d) = \mu_i \cdot d$
- Recherche d'une solution rationnelle.
  - Soit
    $$D(P_1,\ldots,P_p) = 1/\left(\sum_{i=1}^{p} \frac{1}{\lambda_i+\mu_i} \cdot \prod_{j=1}^{i-1} \frac{\mu_j}{\lambda_j+\mu_j}\right).$$
  - Il existe une solution rationnelle optimale où tous les processeurs travaillent si et seulement si
    $$\forall i \in [1, p-1], \quad \lambda_i \leq D(P_{i+1},\ldots,P_p).$$
  - Dans ce cas, dans une solution optimale, tous les processeurs terminent en même temps et :
    $$T = d \cdot D(P_1,\ldots,P_p)\,; \qquad d_i = \frac{1}{\lambda_i+\mu_i} \cdot \left(\prod_{j=1}^{i-1} \frac{\mu_j}{\lambda_j+\mu_j}\right) \cdot T.$$

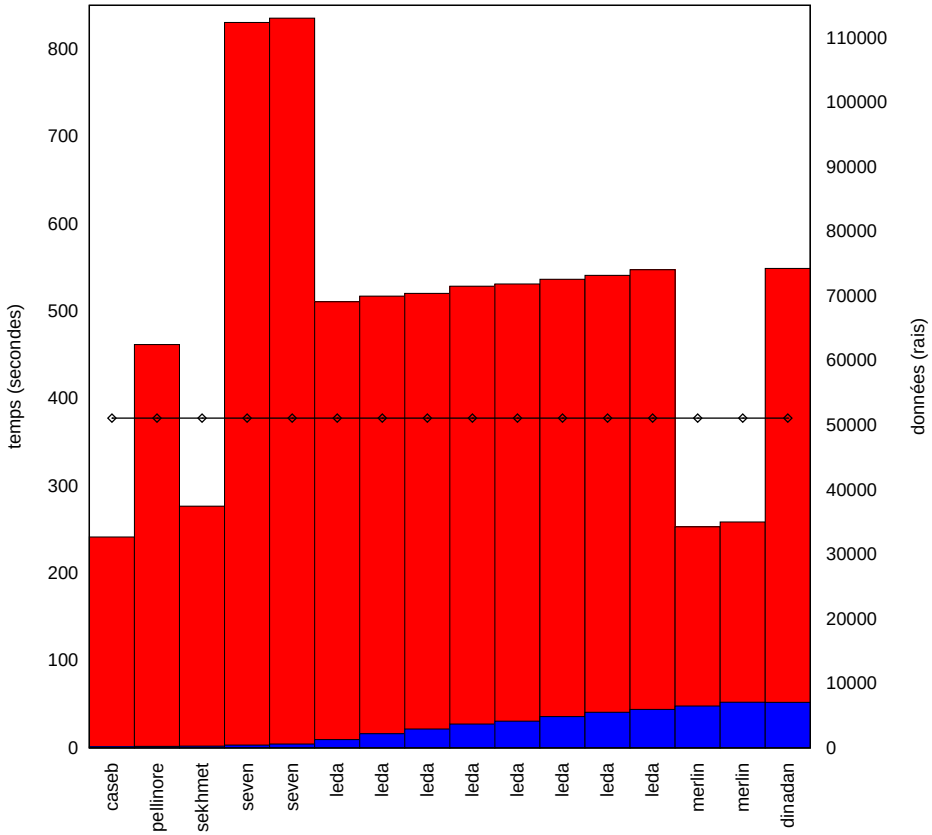
Ordre des processeurs

- Avec une solution *rationnelle*, les processeurs doivent être ordonnés par bandes passantes décroissantes.

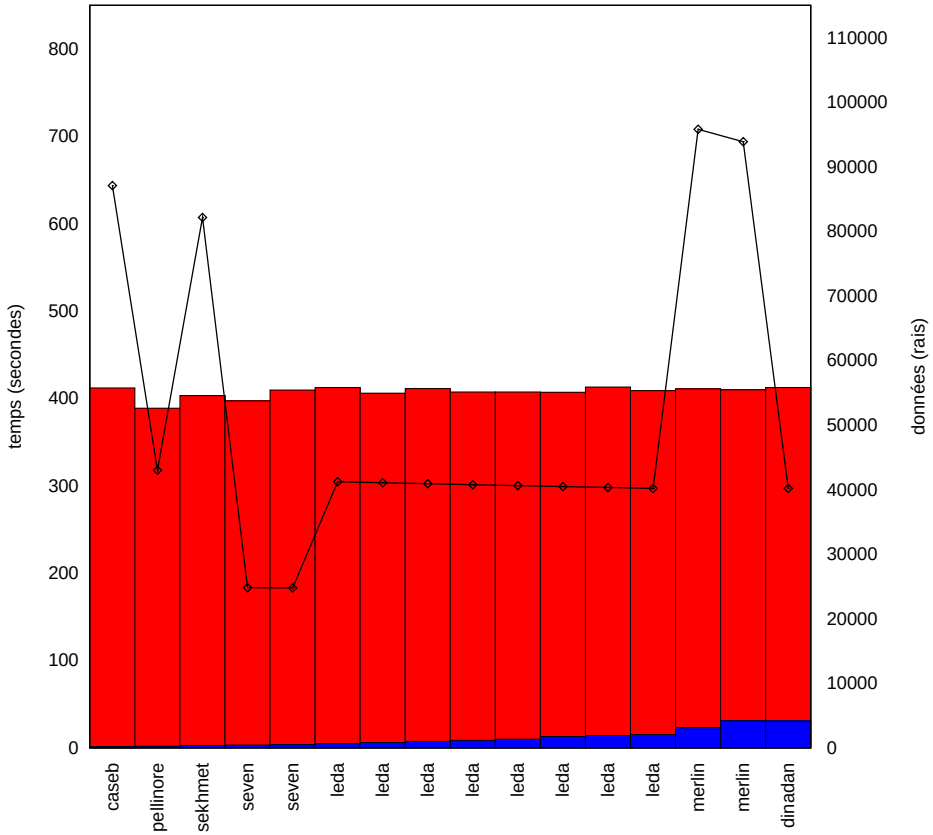
- Avec une solution *entière* (arrondi de la solution rationnelle), cet ordre est garanti :

$$T_{opt} \leq T' \leq T_{opt} + \sum_{j=1}^{p} T_{comm}(j,1) + \max_{1 \leq i \leq p} T_{calc}(i,1)$$

Résultats expérimentaux
Équilibrage

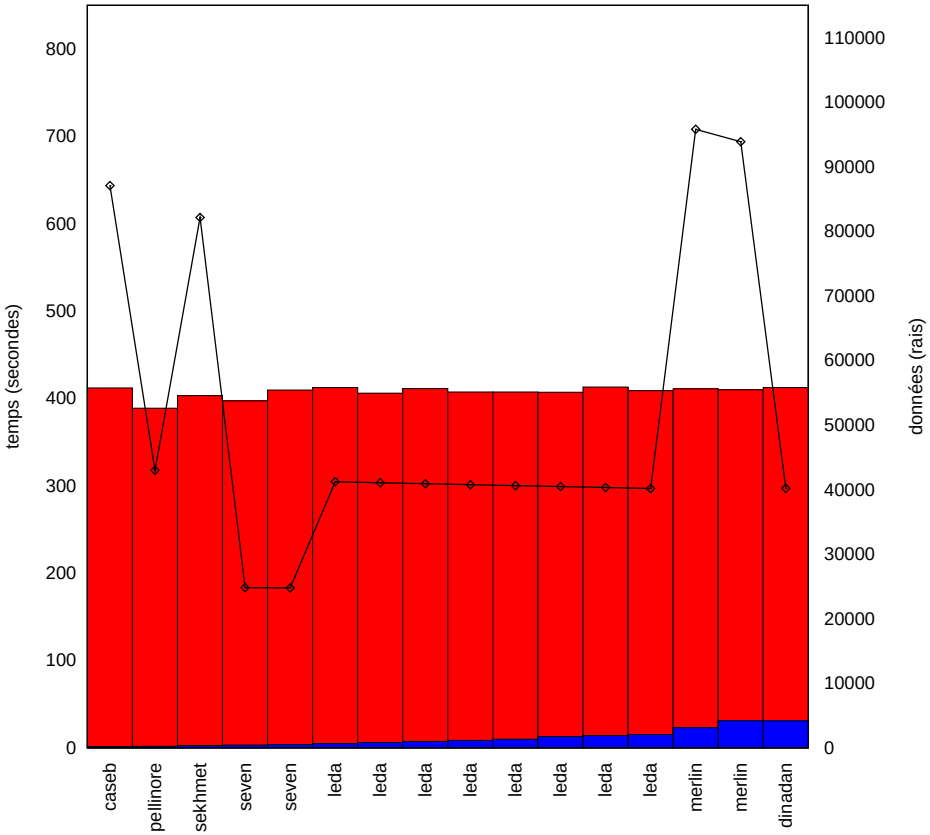


Distribution uniforme.

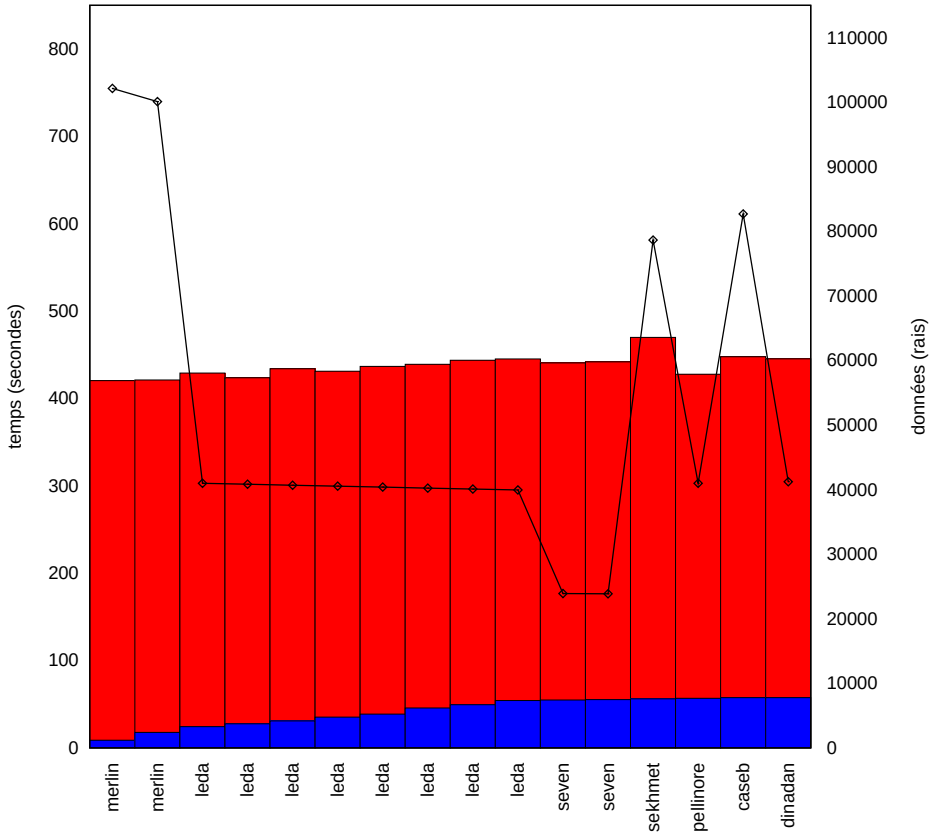


Distribution optimisée.

Résultats expérimentaux
Ordre des processeurs



Bandes passantes décroissantes.



Bandes passantes croissantes.

Conclusion

- Applications maître-esclave sur plates-formes hétérogènes.
- Plusieurs solutions pour équilibrer la charge des processeurs.
- Politique pour ordonner les processeurs de manière optimale.
- Expérimentations en grandeur nature.

À suivre. . .

Généralisation :

- du modèle d'application ;
- du modèle de plate-forme.

Conclusion

- Applications maître-esclave sur plates-formes hétérogènes.
- Plusieurs solutions pour équilibrer la charge des processeurs.
- Politique pour ordonner les processeurs de manière optimale.
- Expérimentations en grandeur nature.

À suivre. . .

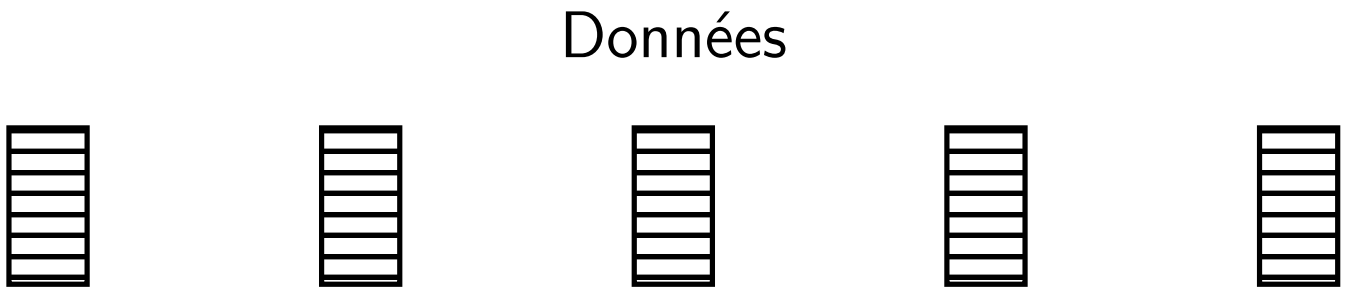
Généralisation :

- du modèle d'application ;
- du modèle de plate-forme.

Données partagées et plusieurs serveurs

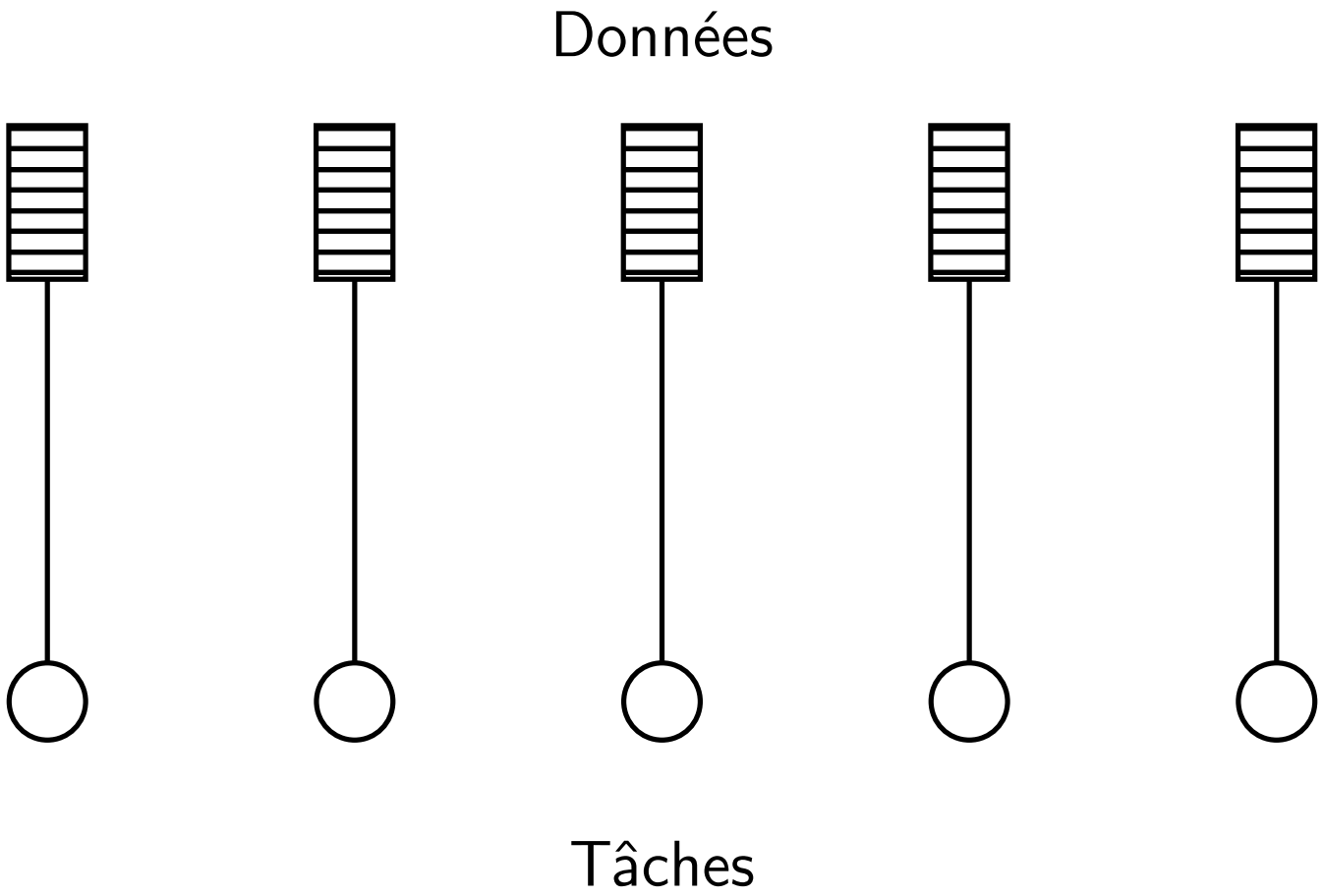
Tâches et données

- Des données de même taille.



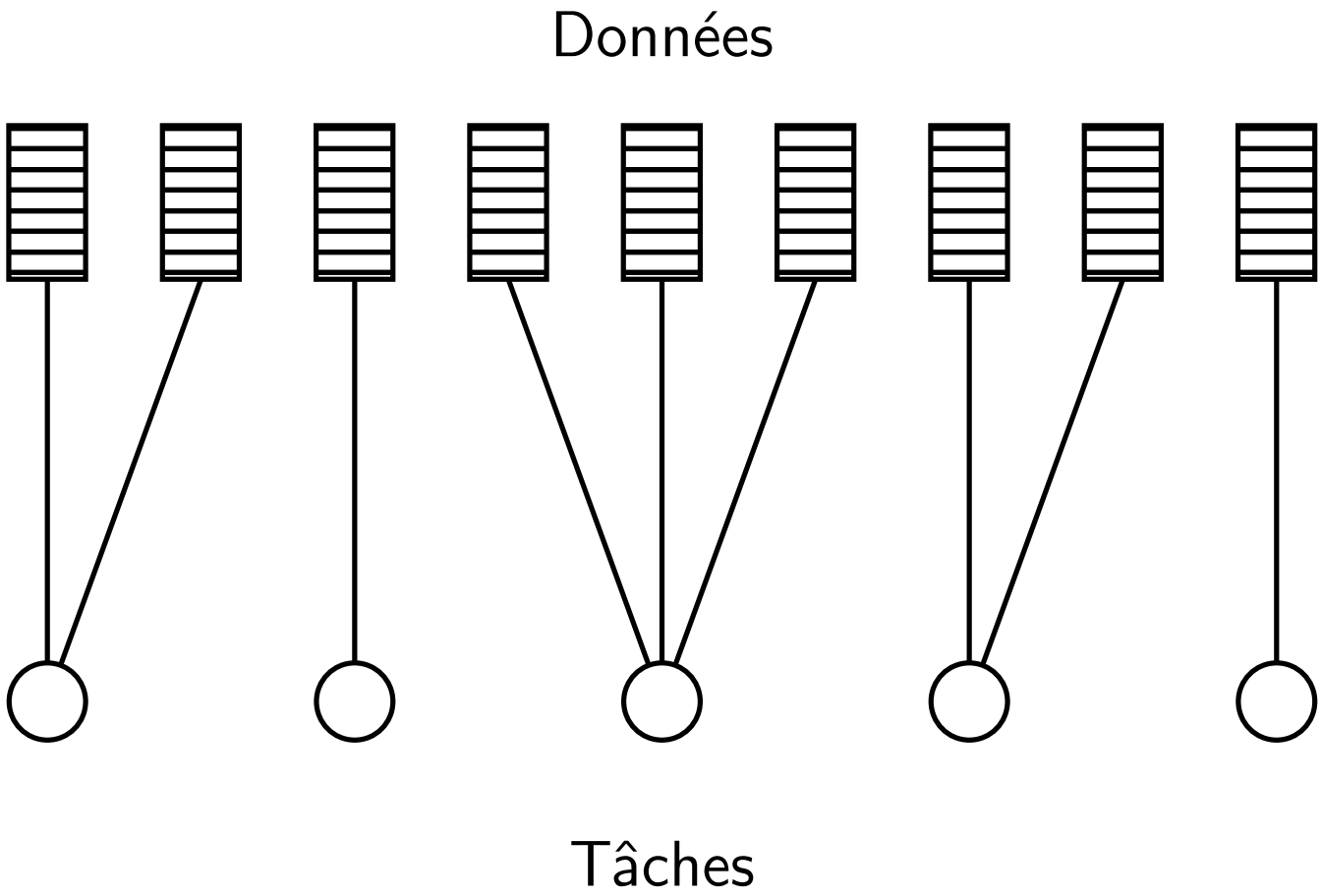
Tâches et données

- À chaque donnée correspond une tâche.



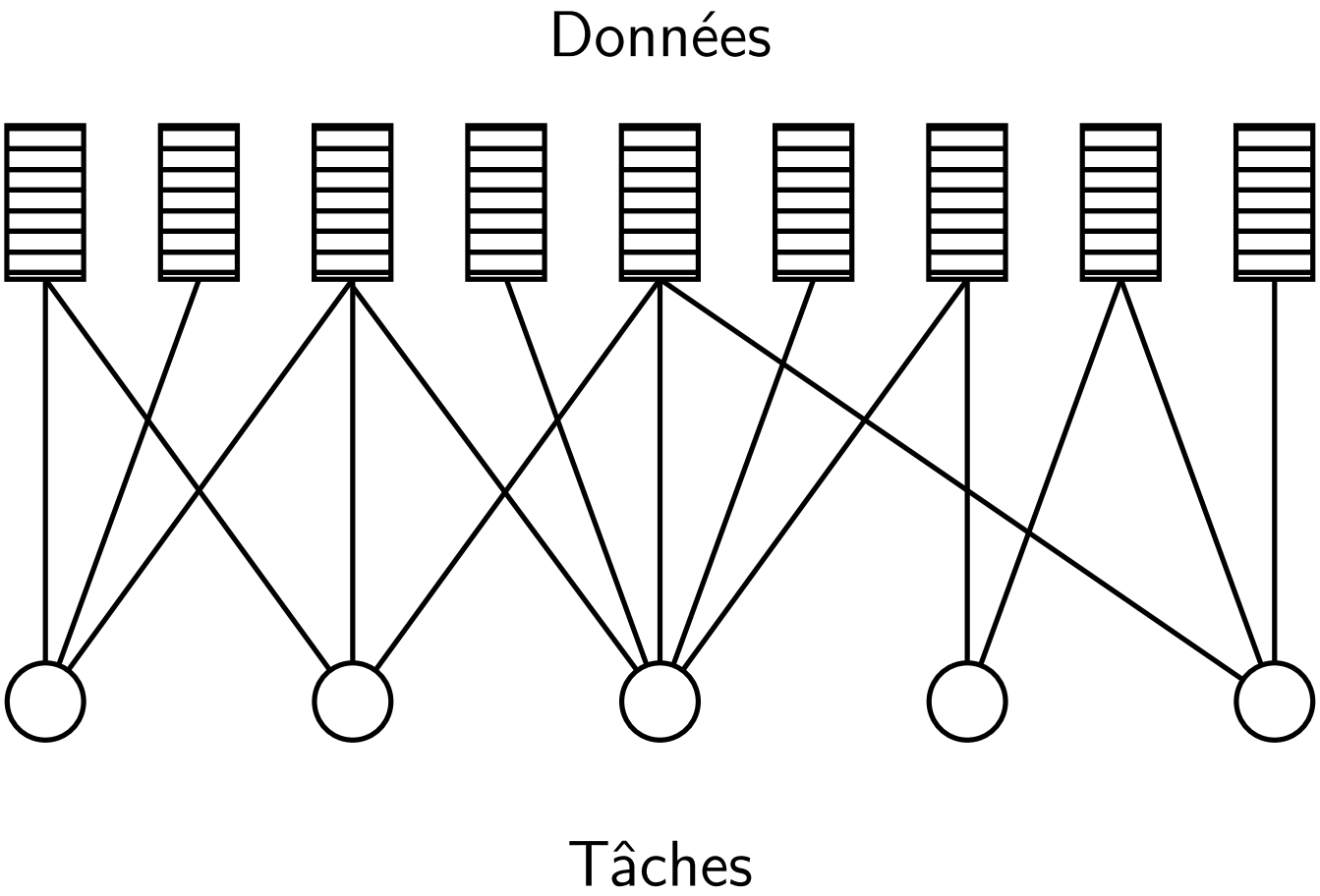
Tâches et données

- Chaque tâche peut dépendre de plusieurs données.



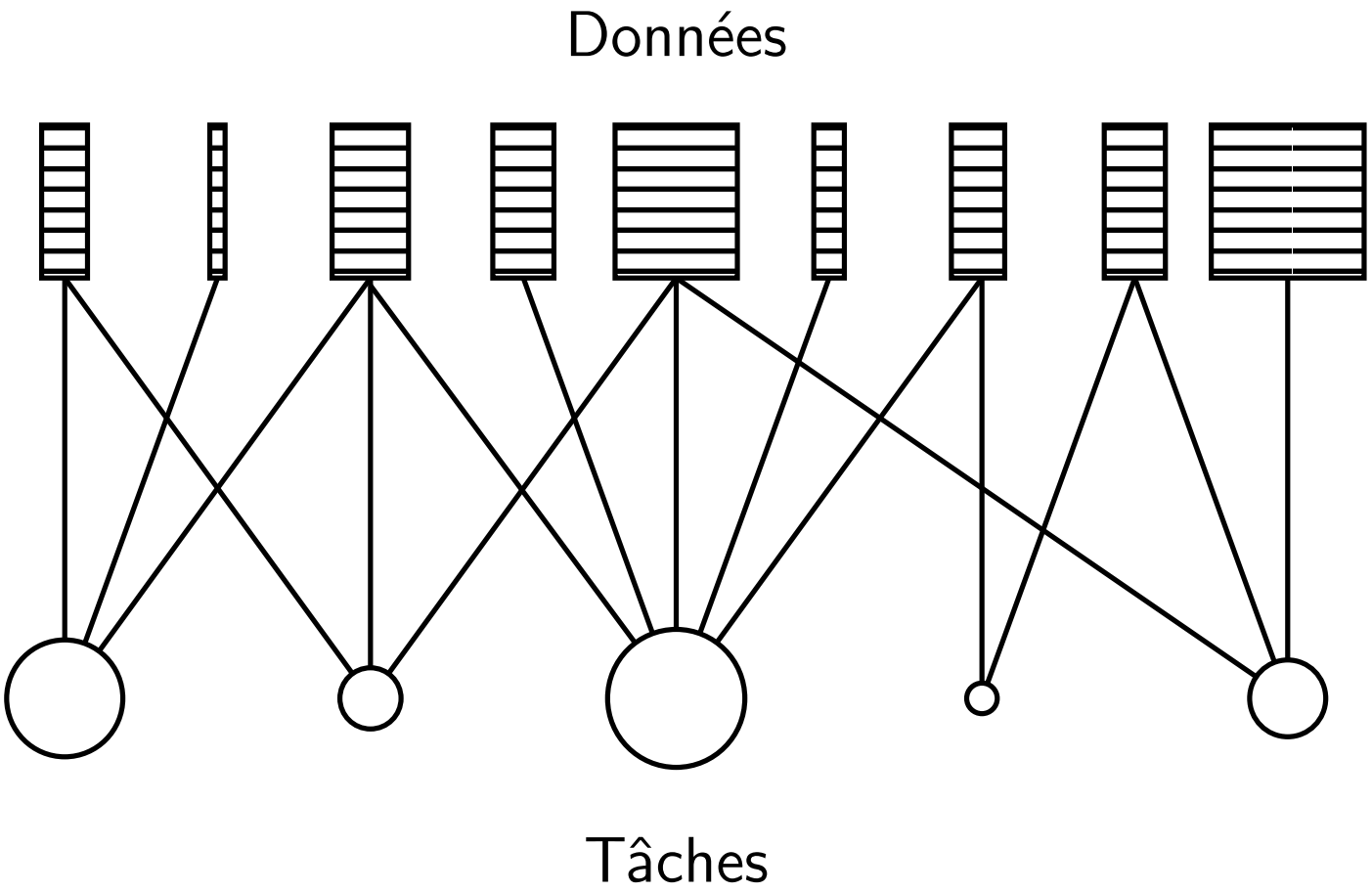
Tâches et données

- Chaque donnée peut servir à plusieurs tâches.



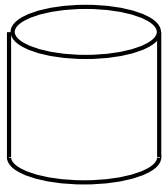
Tâches et données

- Les tailles des tâches et des données sont hétérogènes.



Graphe de plate-forme

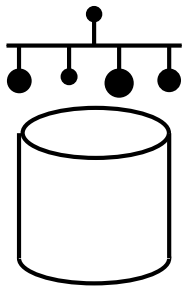
- Un entrepôt pour stocker les données.

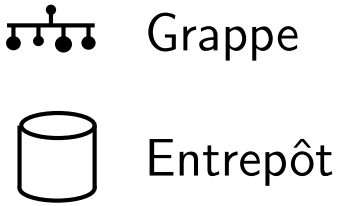


Entrepôt

Graphe de plate-forme

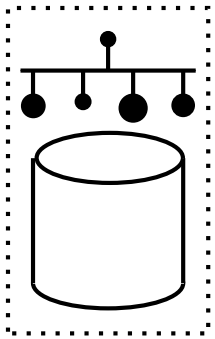
- Une grappe de processeurs (*cluster*) pour exécuter les tâches.

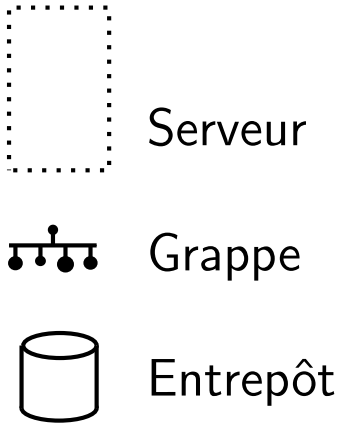




Graphe de plate-forme

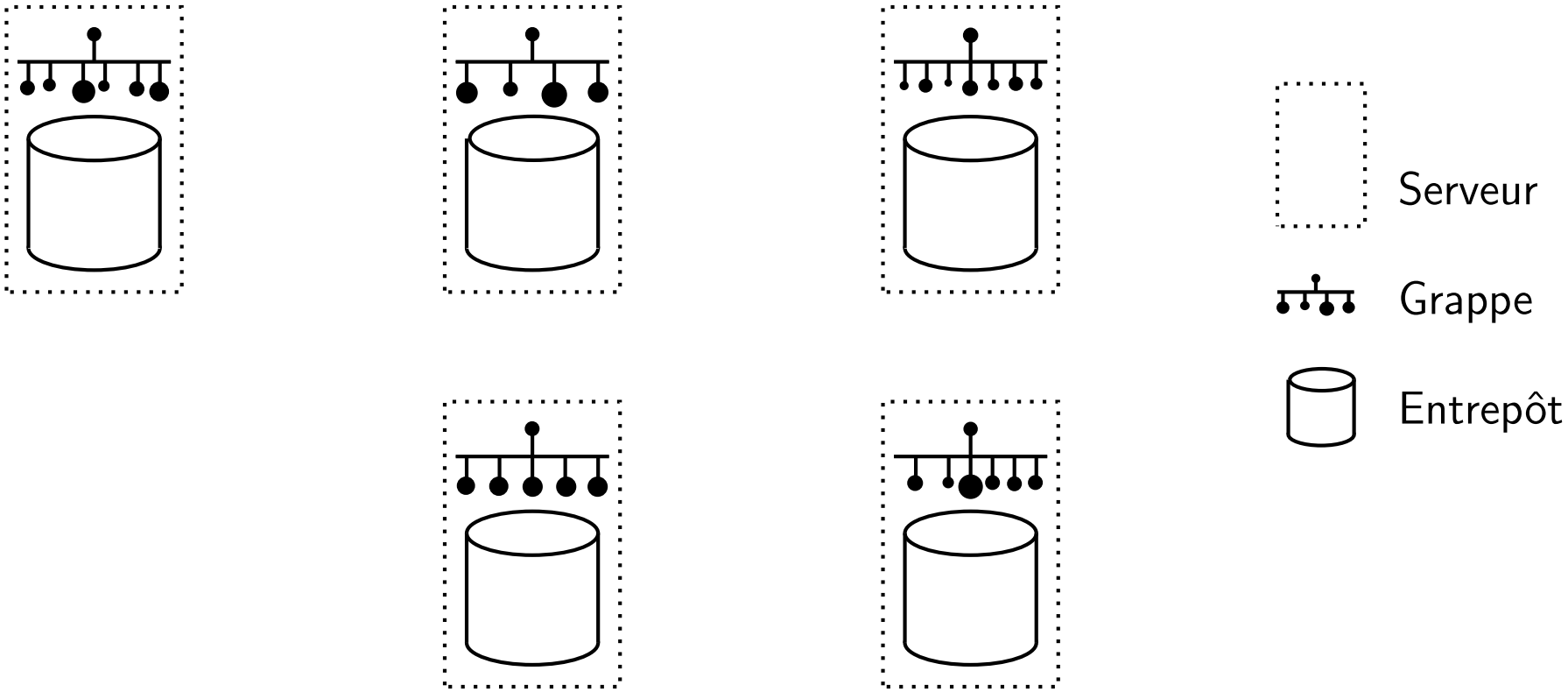
- entrepôt + grappe = serveur





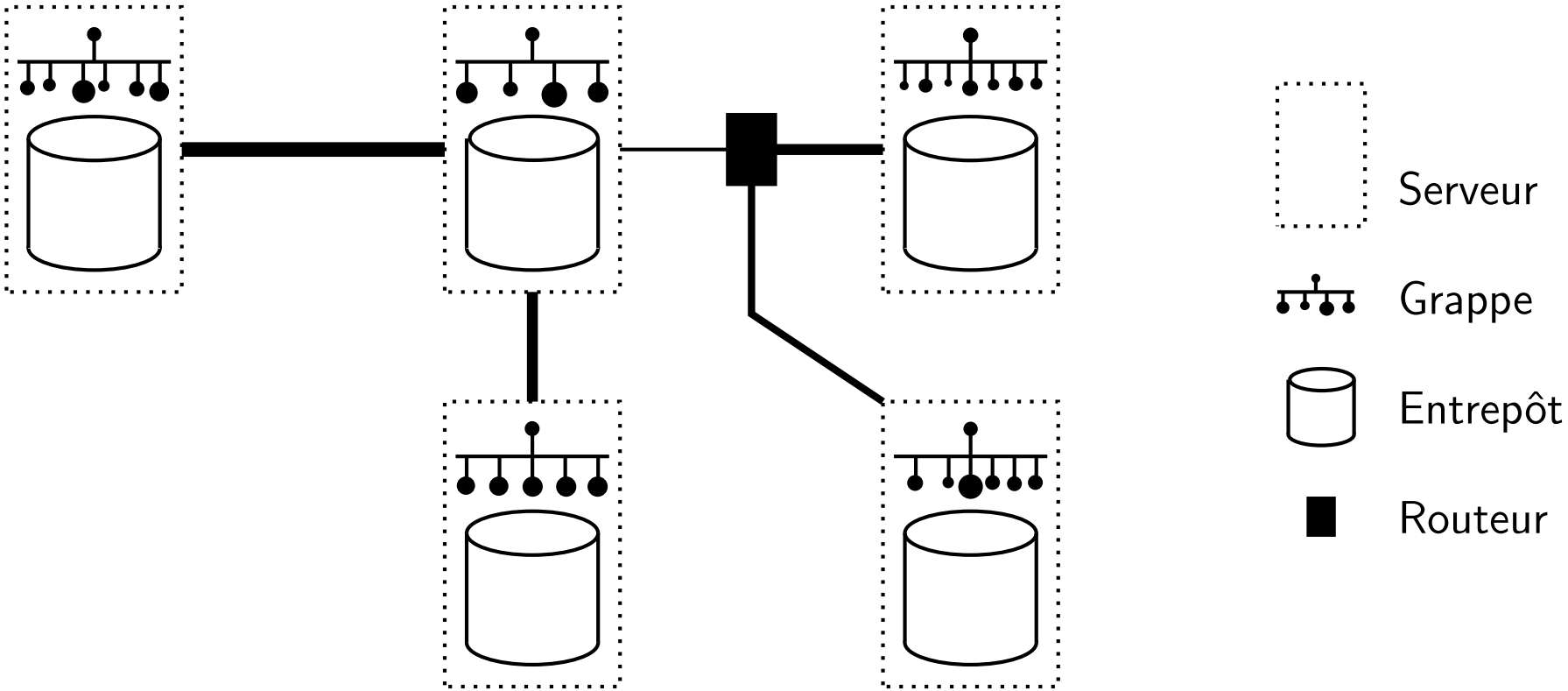
Graphe de plate-forme

- Plusieurs serveurs différents.



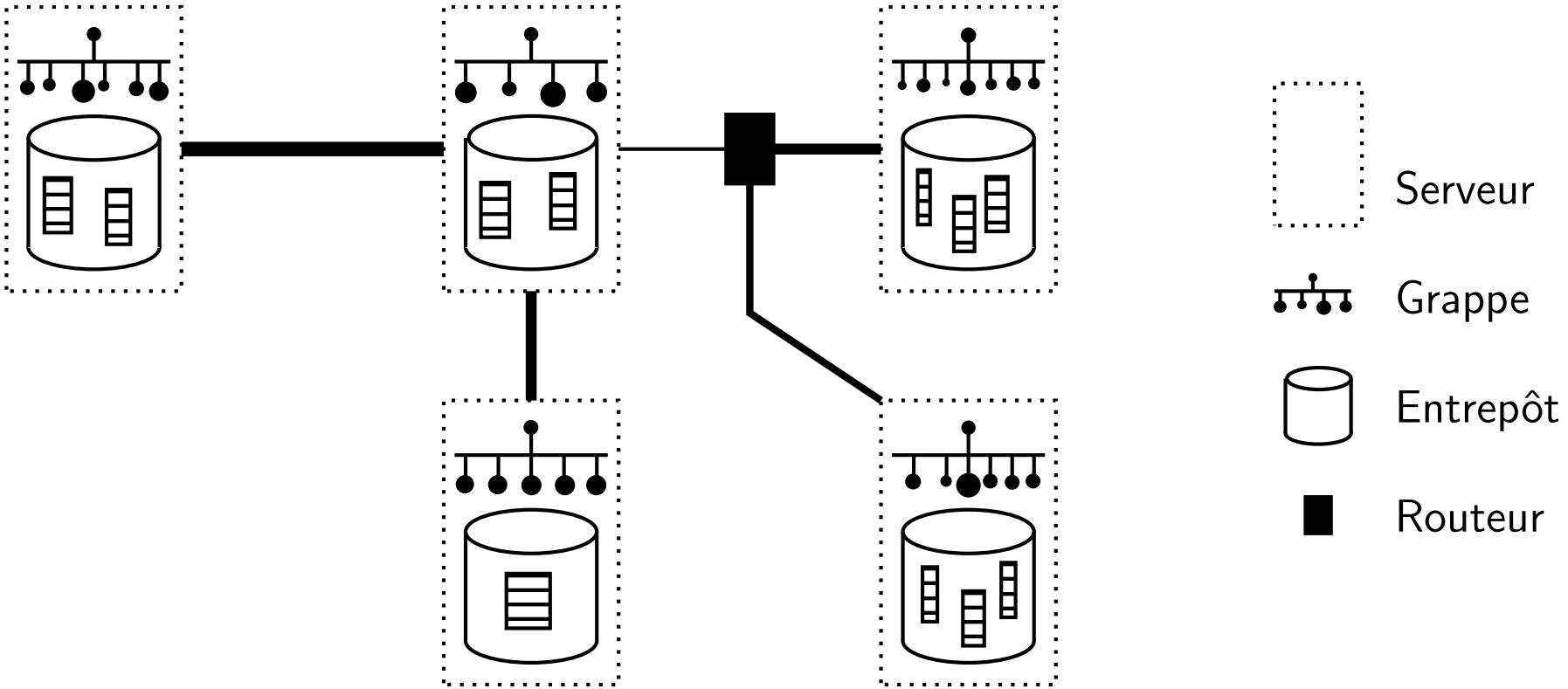
Graphe de plate-forme

- Les serveurs sont reliés entre eux par un réseau d'interconnexion.



Graphe de plate-forme

- Les données sont initialement réparties dans les différents entrepôts.



Objectif

- Minimiser le temps total d'exécution (*makespan*) de l'ensemble des tâches.
- L'exécution est terminée lorsque la dernière tâche est terminée.
- Il faut décider :
  - quelle tâche est à exécuter par chacun des processeurs ;
  - comment sont transférées les données d'un serveur à l'autre ;
  - dans quel ordre.

Hypothèses

- Communications suivent le modèle un-port.
- Le routage au sein du graphe de plate-forme est statique.
- Une donnée déposée dans un entrepôt y reste disponible pour le reste de l'ordonnancement (*persistance des données*).
- Lors du transfert d'une donnée, une copie est déposée dans l'entrepôt de chaque serveur intermédiaire (*store and forward*).

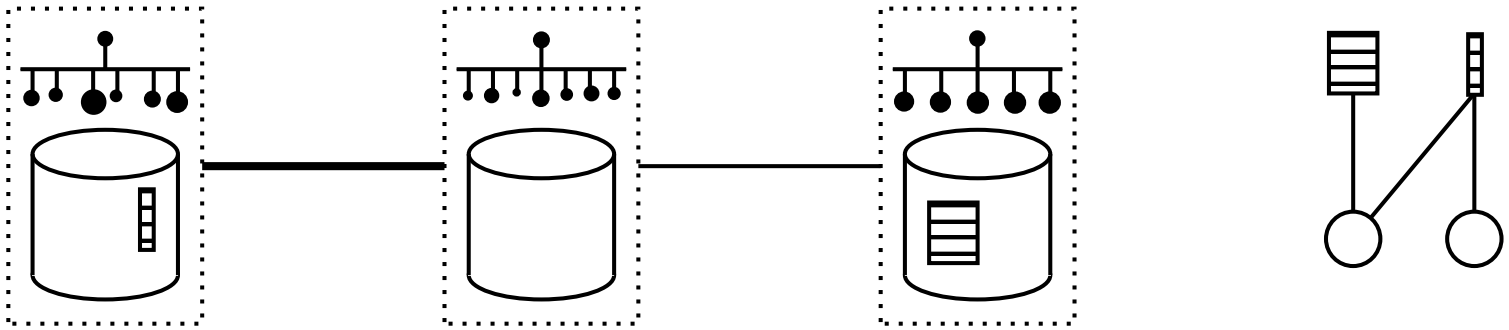
Hypothèses

- Communications suivent le modèle un-port.
- Le routage au sein du graphe de plate-forme est statique.

- Une donnée déposée dans un entrepôt y reste disponible pour le reste de l'ordonnancement (*persistance des données*).
- Lors du transfert d'une donnée, une copie est déposée dans l'entrepôt de chaque serveur intermédiaire (*store and forward*).

Hypothèses

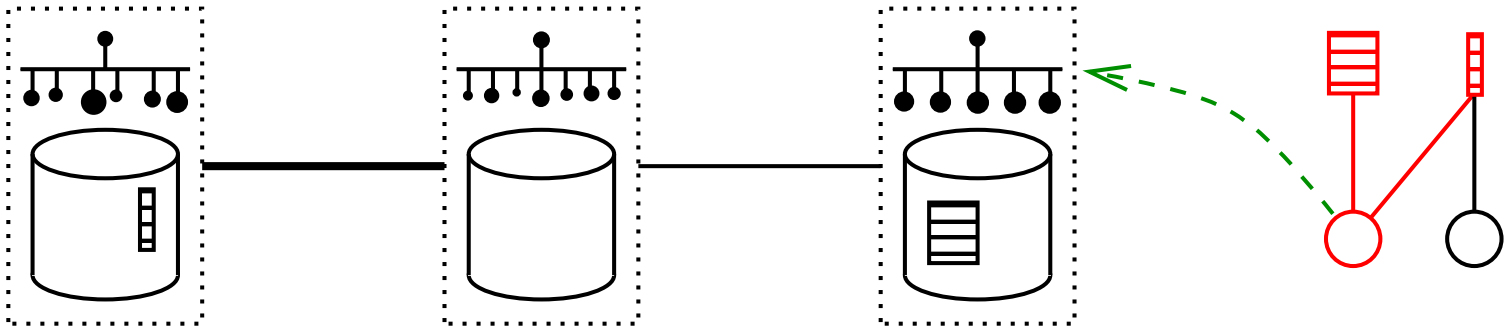
- Communications suivent le modèle un-port.
- Le routage au sein du graphe de plate-forme est statique.



- Une donnée déposée dans un entrepôt y reste disponible pour le reste de l'ordonnancement (*persistance des données*).
- Lors du transfert d'une donnée, une copie est déposée dans l'entrepôt de chaque serveur intermédiaire (*store and forward*).

Hypothèses

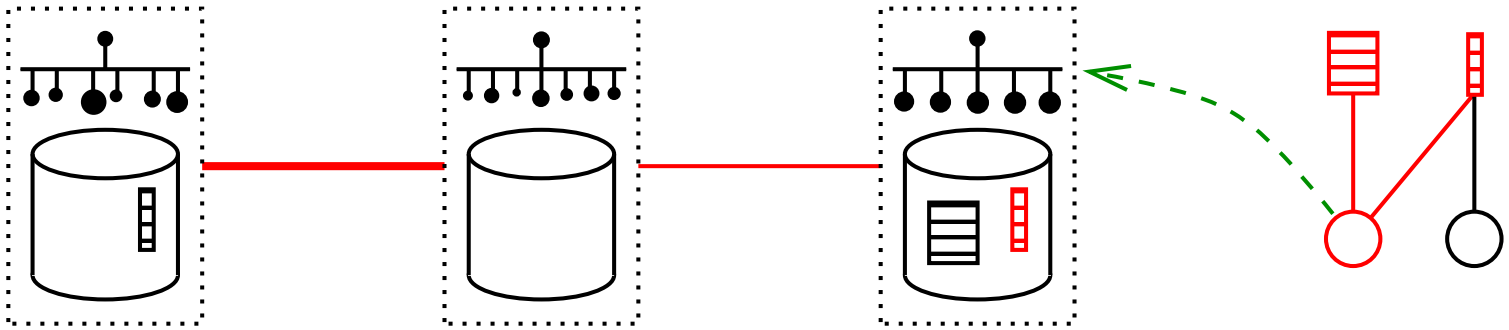
- Communications suivent le modèle un-port.
- Le routage au sein du graphe de plate-forme est statique.



- Une donnée déposée dans un entrepôt y reste disponible pour le reste de l'ordonnancement (*persistance des données*).
- Lors du transfert d'une donnée, une copie est déposée dans l'entrepôt de chaque serveur intermédiaire (*store and forward*).

Hypothèses

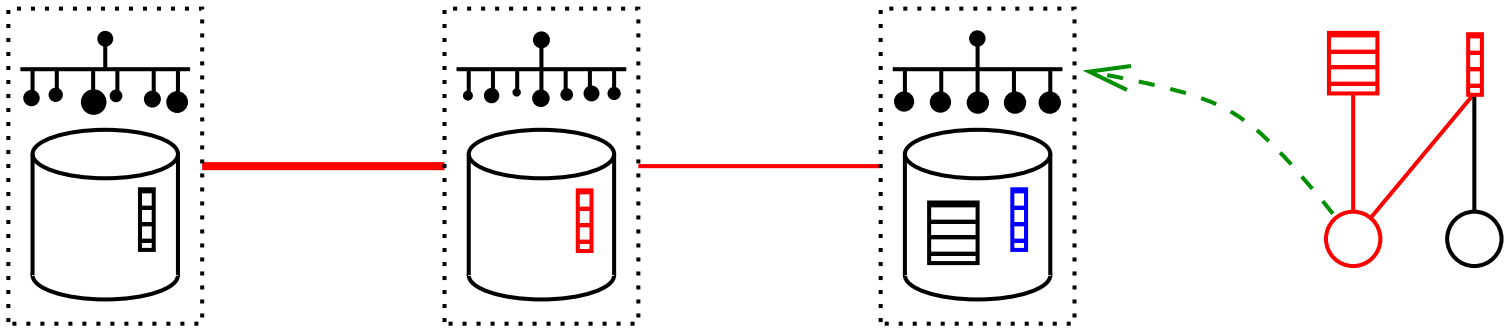
- Communications suivent le modèle un-port.
- Le routage au sein du graphe de plate-forme est statique.



- Une donnée déposée dans un entrepôt y reste disponible pour le reste de l'ordonnancement (*persistance des données*).
- Lors du transfert d'une donnée, une copie est déposée dans l'entrepôt de chaque serveur intermédiaire (*store and forward*).

Hypothèses

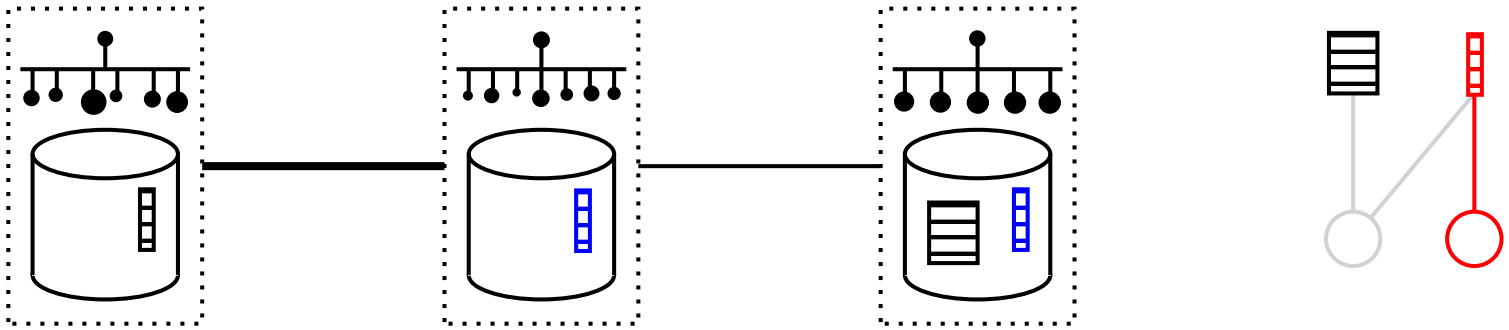
- Communications suivent le modèle un-port.
- Le routage au sein du graphe de plate-forme est statique.



- Une donnée déposée dans un entrepôt y reste disponible pour le reste de l'ordonnancement (*persistance des données*).
- Lors du transfert d'une donnée, une copie est déposée dans l'entrepôt de chaque serveur intermédiaire (*store and forward*).

Hypothèses

- Communications suivent le modèle un-port.
- Le routage au sein du graphe de plate-forme est statique.



- Une donnée déposée dans un entrepôt y reste disponible pour le reste de l'ordonnancement (*persistance des données*).
- Lors du transfert d'une donnée, une copie est déposée dans l'entrepôt de chaque serveur intermédiaire (*store and forward*).

Complexité
Définitions

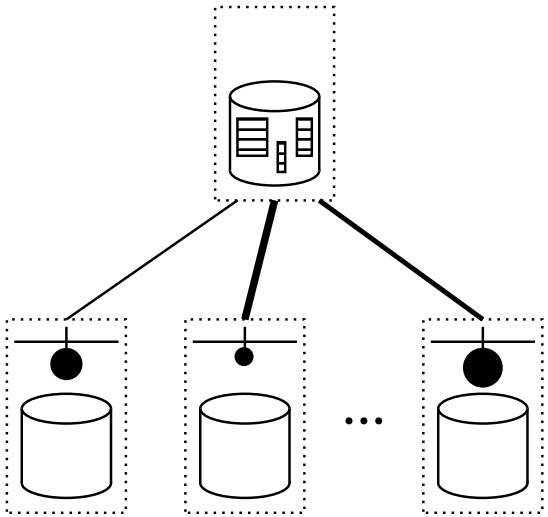
- Les problèmes sont classés en fonction de leur difficulté intrinsèque.
- Dans la suite, on distingue deux classes de problèmes :

  polynomial : il existe un algorithme pour résoudre le problème en temps polynomial en fonction de la taille du problème ;

  NP-complet : il n'existe pas d'algorithme pour résoudre le problème en temps polynomial en fonction de la taille du problème (sauf si $P = NP$).

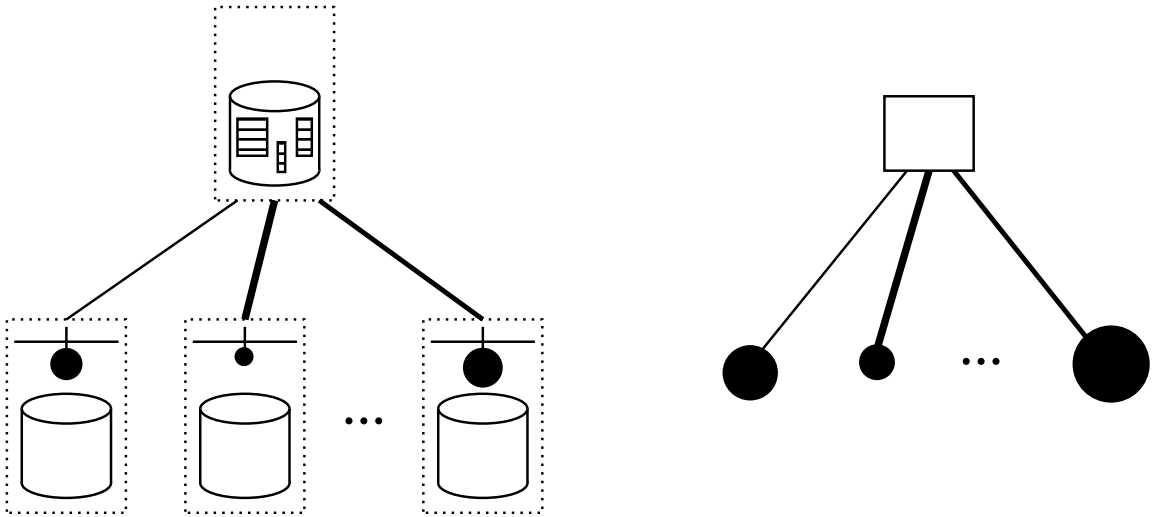
Complexité
Le cas maître-esclave

- Cas particulier du graphe de plate-forme : maître-esclave.



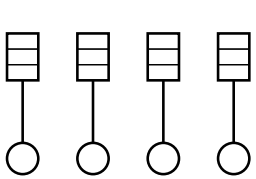
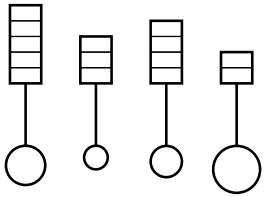
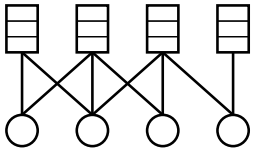
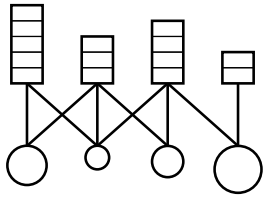
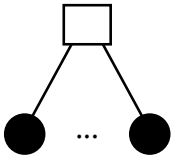
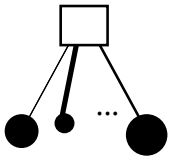
Complexité
Le cas maître-esclave

- Cas particulier du graphe de plate-forme : maître-esclave.



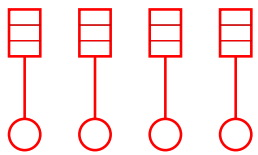
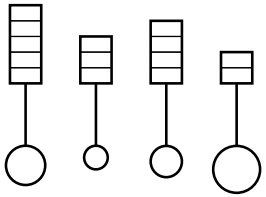
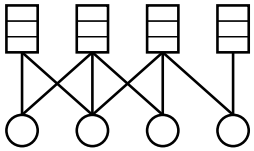
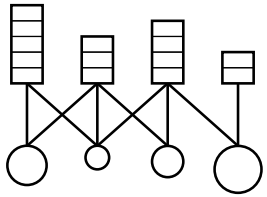
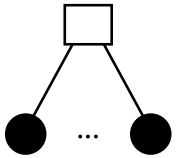
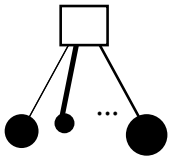
Résultats de complexité

Plate-forme maître-esclave

				
	polynomial	polynomial	?	NP-complet
	polynomial	NP-complet	?	NP-complet
	polynomial	NP-complet	NP-complet	NP-complet

Résultats de complexité

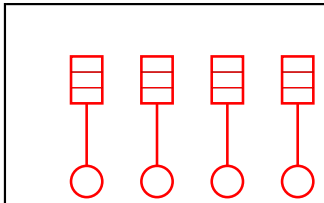
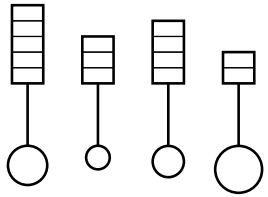
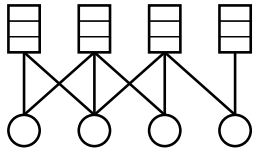
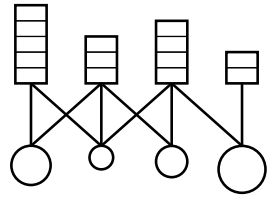
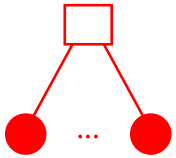
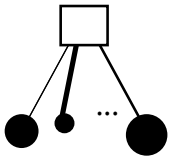
Plate-forme maître-esclave

				
	polynomial	polynomial	?	NP-complet
	polynomial	NP-complet	?	NP-complet
	polynomial	NP-complet	NP-complet	NP-complet

- Trivial.

Résultats de complexité

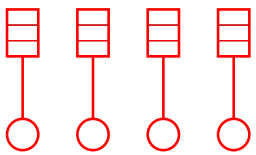
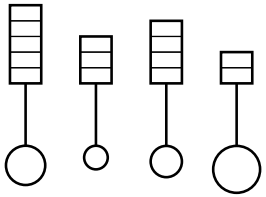
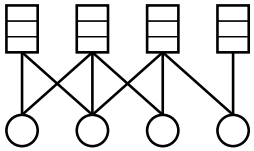
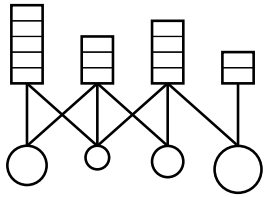
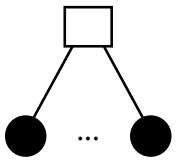
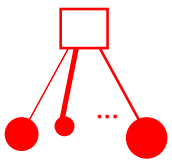
Plate-forme maître-esclave

				
	polynomial	polynomial	?	NP-complet
	polynomial	NP-complet	?	NP-complet
	polynomial	NP-complet	NP-complet	NP-complet

- Algorithme du tourniquet (*round-robin*).

Résultats de complexité

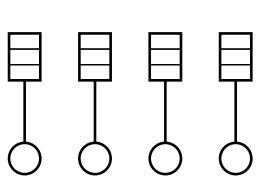
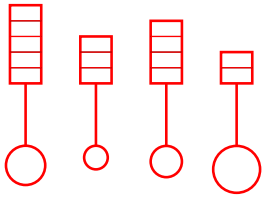
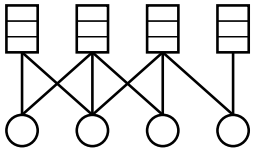
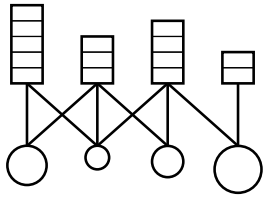
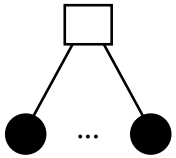
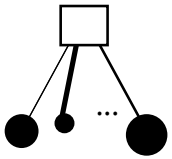
Plate-forme maître-esclave

				
	polynomial	polynomial	?	NP-complet
	polynomial	NP-complet	?	NP-complet
	polynomial	NP-complet	NP-complet	NP-complet

- Algorithme de Beaumont, Legrand et Robert (2002).

Résultats de complexité

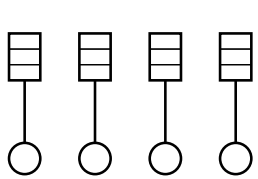
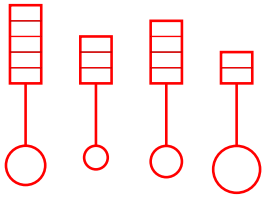
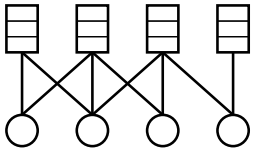
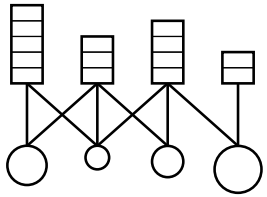
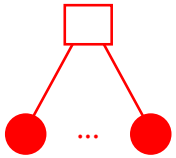
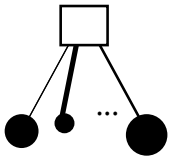
Plate-forme maître-esclave

				
	polynomial	polynomial	?	NP-complet
	polynomial	NP-complet	?	NP-complet
	polynomial	NP-complet	NP-complet	NP-complet

- Algorithme de Johnson (1954) pour *flow-shop* à deux machines.

Résultats de complexité

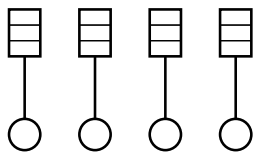
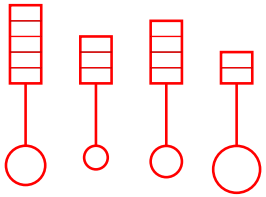
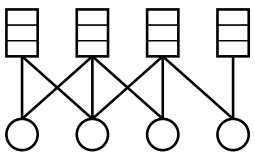
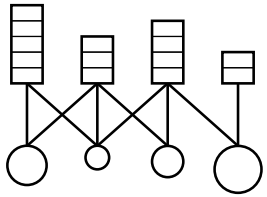
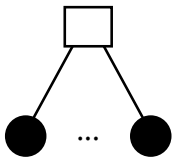
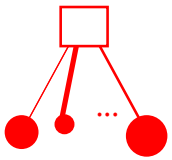
Plate-forme maître-esclave

				
	polynomial	polynomial	?	NP-complet
	polynomial	NP-complet	?	NP-complet
	polynomial	NP-complet	NP-complet	NP-complet

- 2-PARTITION.

Résultats de complexité

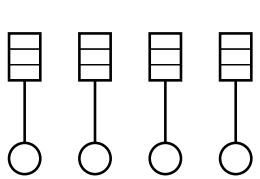
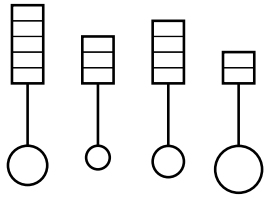
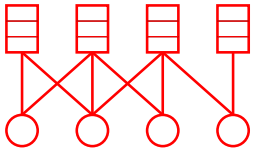
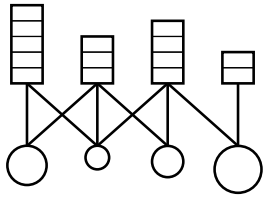
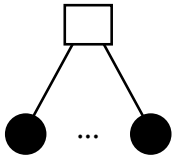
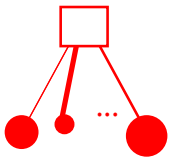
Plate-forme maître-esclave

				
	polynomial	polynomial	?	NP-complet
	polynomial	NP-complet	?	NP-complet
	polynomial	NP-complet	NP-complet	NP-complet

- 2-Partition.

Résultats de complexité

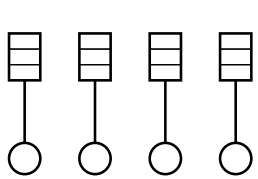
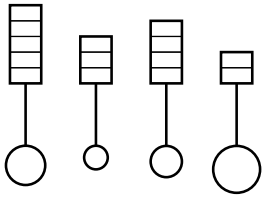
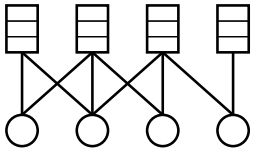
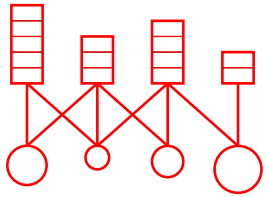
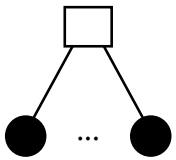
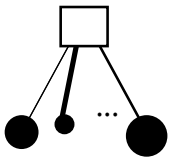
Plate-forme maître-esclave

				
	polynomial	polynomial	?	NP-complet
	polynomial	NP-complet	?	NP-complet
	polynomial	NP-complet	NP-complet	NP-complet

- Démonstration dans la thèse.

Résultats de complexité

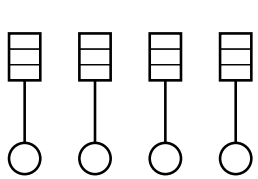
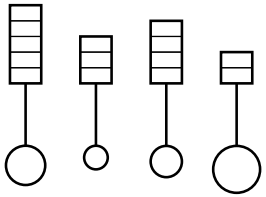
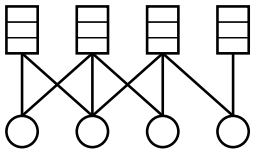
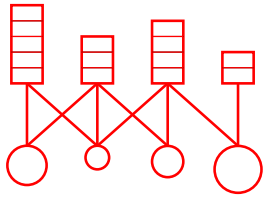
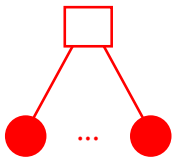
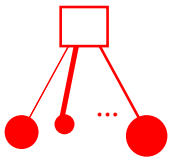
Plate-forme maître-esclave

				
	polynomial	polynomial	?	NP-complet
	polynomial	NP-complet	?	NP-complet
	polynomial	NP-complet	NP-complet	NP-complet

- Démonstration dans la thèse.

Résultats de complexité

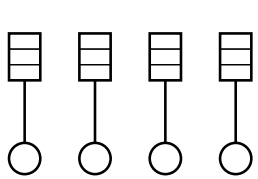
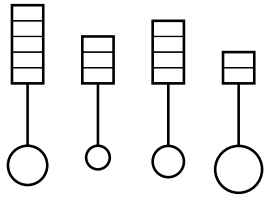
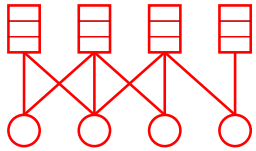
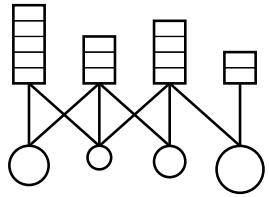
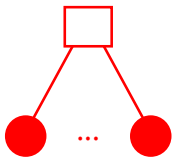
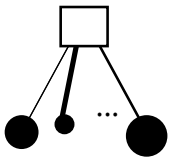
Plate-forme maître-esclave

				
	polynomial	polynomial	?	NP-complet
	polynomial	NP-complet	?	NP-complet
	polynomial	NP-complet	NP-complet $\Rightarrow$	NP-complet

- Généralisation des résultats précédents.

Résultats de complexité

Plate-forme maître-esclave

				
	polynomial	polynomial	?	NP-complet
	polynomial	NP-complet	?	NP-complet
	polynomial	NP-complet	NP-complet	NP-complet

- Complexité à ce jour inconnue.

Résultats de complexité
Plate-forme distribuée

- Même dans une version entièrement homogène, où
  - le graphe de plate-forme est quelconque ;
  - toutes les données ont la même taille ;
  - tous les liens de communications ont la même bande passante ;
  - le temps d'exécution des tâches est nul ;

  le problème est NP-complet.
- Dans la version générale du problème, il y a des poids pour les tâches et pour les données, et la plate-forme est hétérogène.
- Conception d'heuristiques polynomiales pour résoudre le problème.

Résultats de complexité
Plate-forme distribuée

- Même dans une version entièrement homogène, où
  - le graphe de plate-forme est quelconque ;
  - toutes les données ont la même taille ;
  - tous les liens de communications ont la même bande passante ;
  - le temps d'exécution des tâches est nul ;

  le problème est NP-complet.

- Dans la version générale du problème, il y a des poids pour les tâches et pour les données, et la plate-forme est hétérogène.

- Conception d'heuristiques polynomiales pour résoudre le problème.

Heuristiques de référence

Casanova, Legrand, Zagorodnov et Berman (HCW 2000)
pour des plates-formes maître-esclave.

min-min

Tant qu'il reste des tâches à ordonnancer, faire

1. pour chaque tâche T_k qui reste à ordonnancer et pour chaque processeur P_i, évaluer le *temps de complétion minimum* (MCT) de T_k si elle est placée sur P_i ;
2. choisir un couple (T_k, P_i) avec un MCT minimum, puis ordonnancer au plus tôt T_k sur P_i.

sufferage

2. choisir la tâche T_k qui serait la plus pénalisée si elle n'était pas placée sur le processeur qui lui est le plus favorable, mais sur le deuxième plus favorable ; ordonnancer T_k sur P_i.

Heuristiques de référence

Casanova, Legrand, Zagorodnov et Berman (HCW 2000)
pour des plates-formes maître-esclave.

min-min

Tant qu'il reste des tâches à ordonnancer, faire

1. pour chaque tâche T_k qui reste à ordonnancer et pour chaque processeur P_i, évaluer le *temps de complétion minimum* (MCT) de T_k si elle est placée sur P_i ;
2. choisir un couple (T_k, P_i) avec un MCT minimum, puis ordonnancer au plus tôt T_k sur P_i.

sufferage

2. choisir la tâche T_k qui serait la plus pénalisée si elle n'était pas placée sur le processeur qui lui est le plus favorable, mais sur le deuxième plus favorable ; ordonnancer T_k sur P_i.

Heuristiques de référence

Casanova, Legrand, Zagorodnov et Berman (HCW 2000)
pour des plates-formes maître-esclave.

min-min

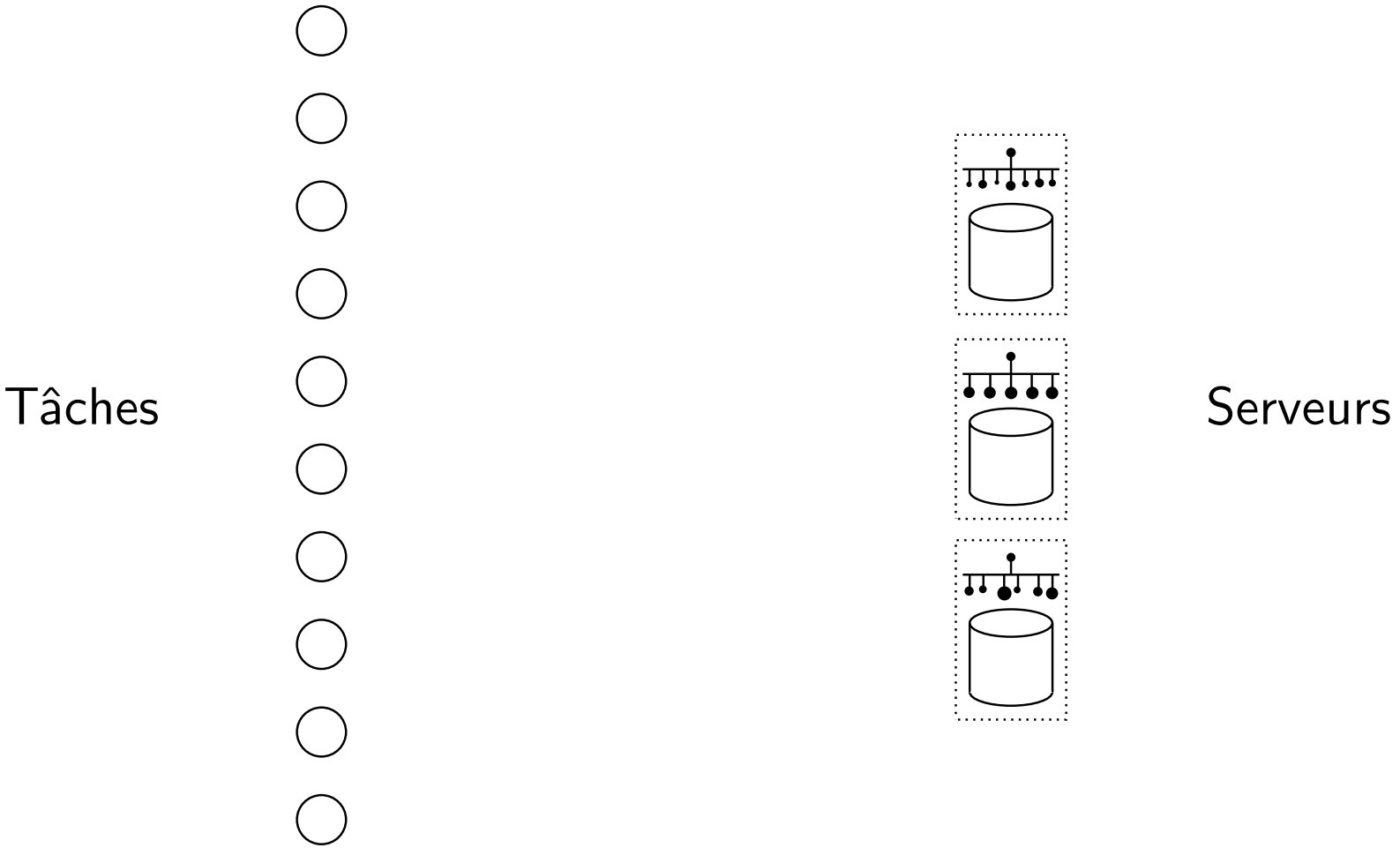
Tant qu'il reste des tâches à ordonnancer, faire

1. pour chaque tâche T_k qui reste à ordonnancer et pour chaque processeur P_i, évaluer le *temps de complétion minimum* (MCT) de T_k si elle est placée sur P_i ;
2. choisir un couple (T_k, P_i) avec un MCT minimum, puis ordonnancer au plus tôt T_k sur P_i.

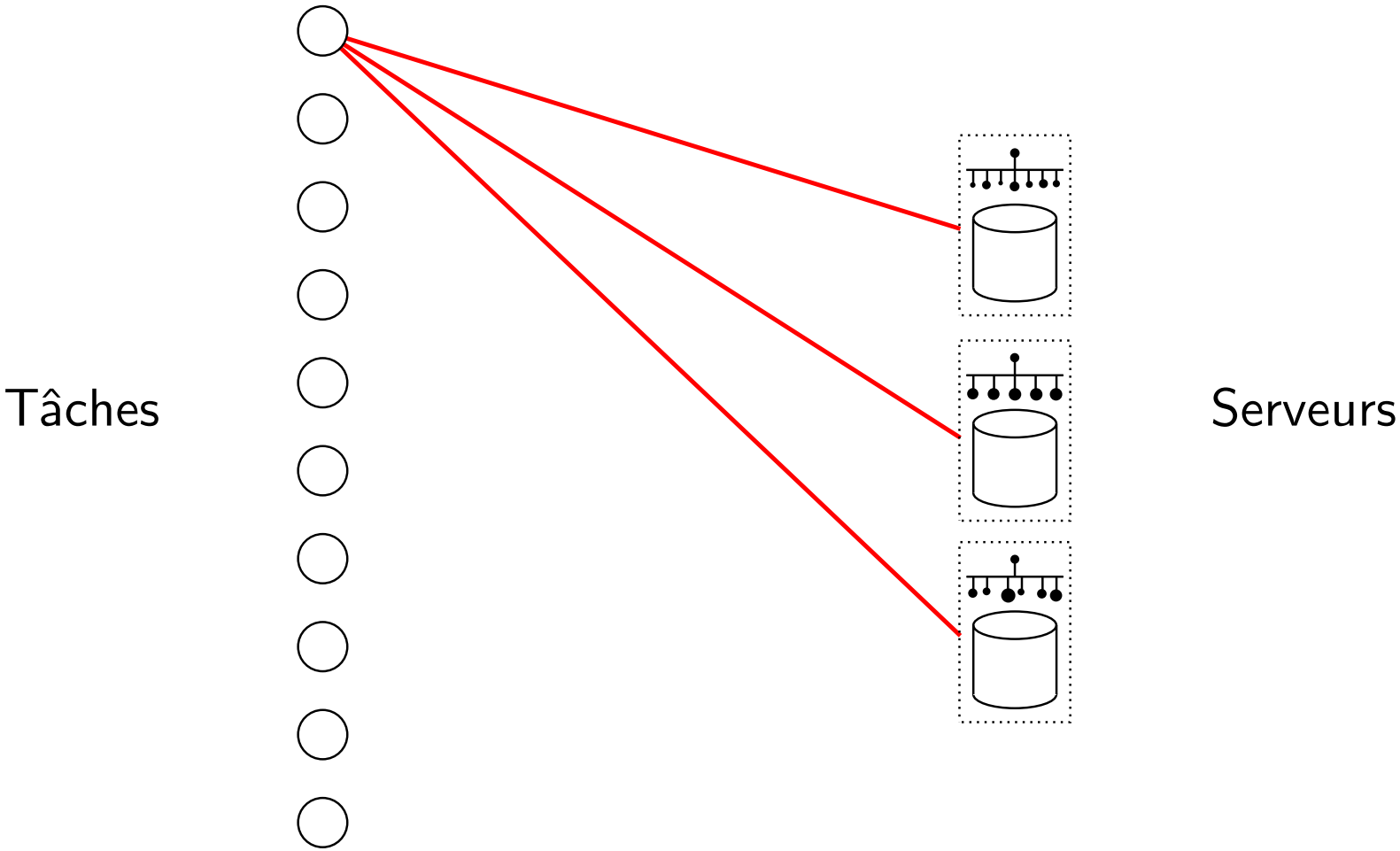
sufferage

2. choisir la tâche T_k qui serait la plus pénalisée si elle n'était pas placée sur le processeur qui lui est le plus favorable, mais sur le deuxième plus favorable ; ordonnancer T_k sur P_i.

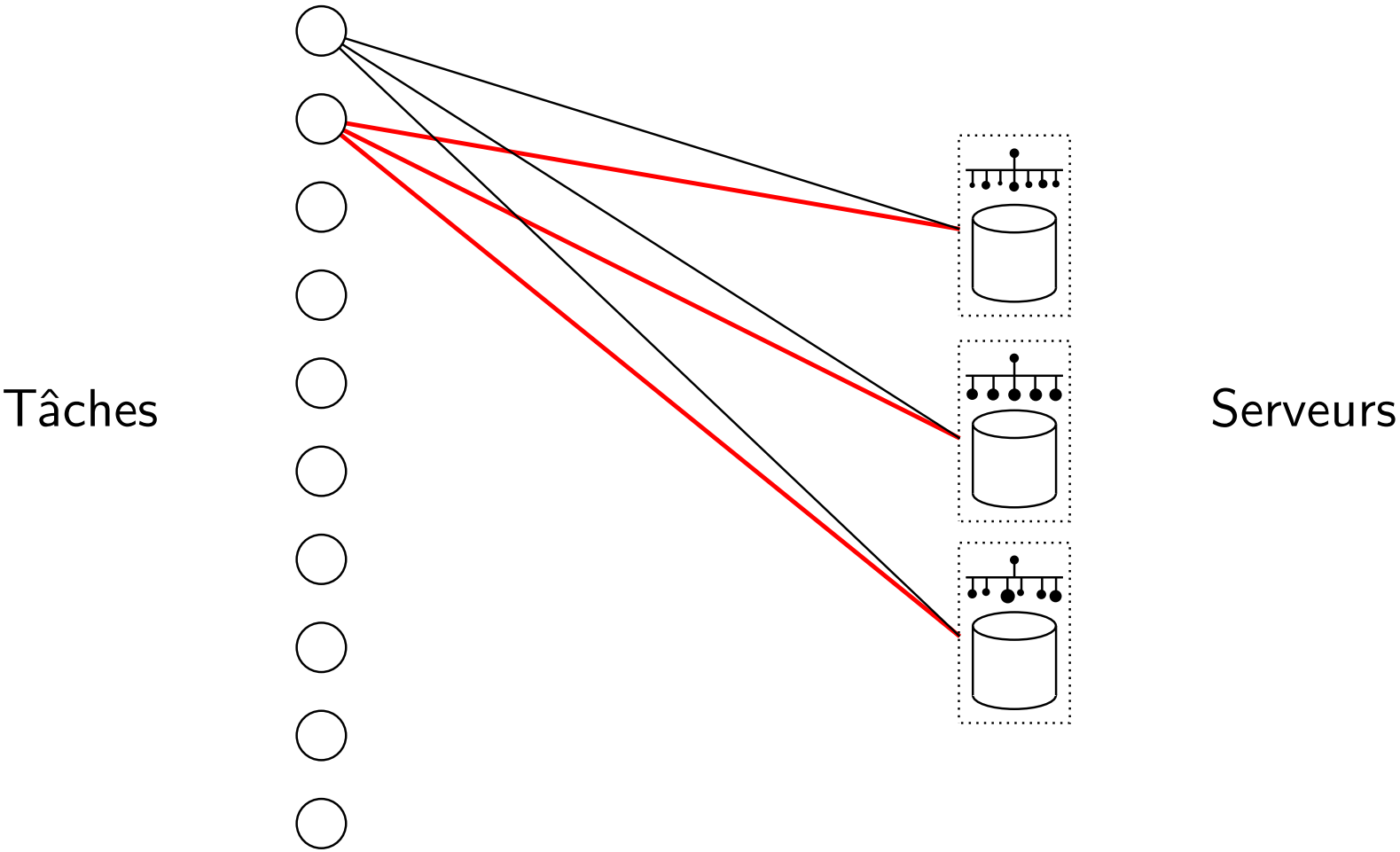
Exemple d'exécution du *min-min*



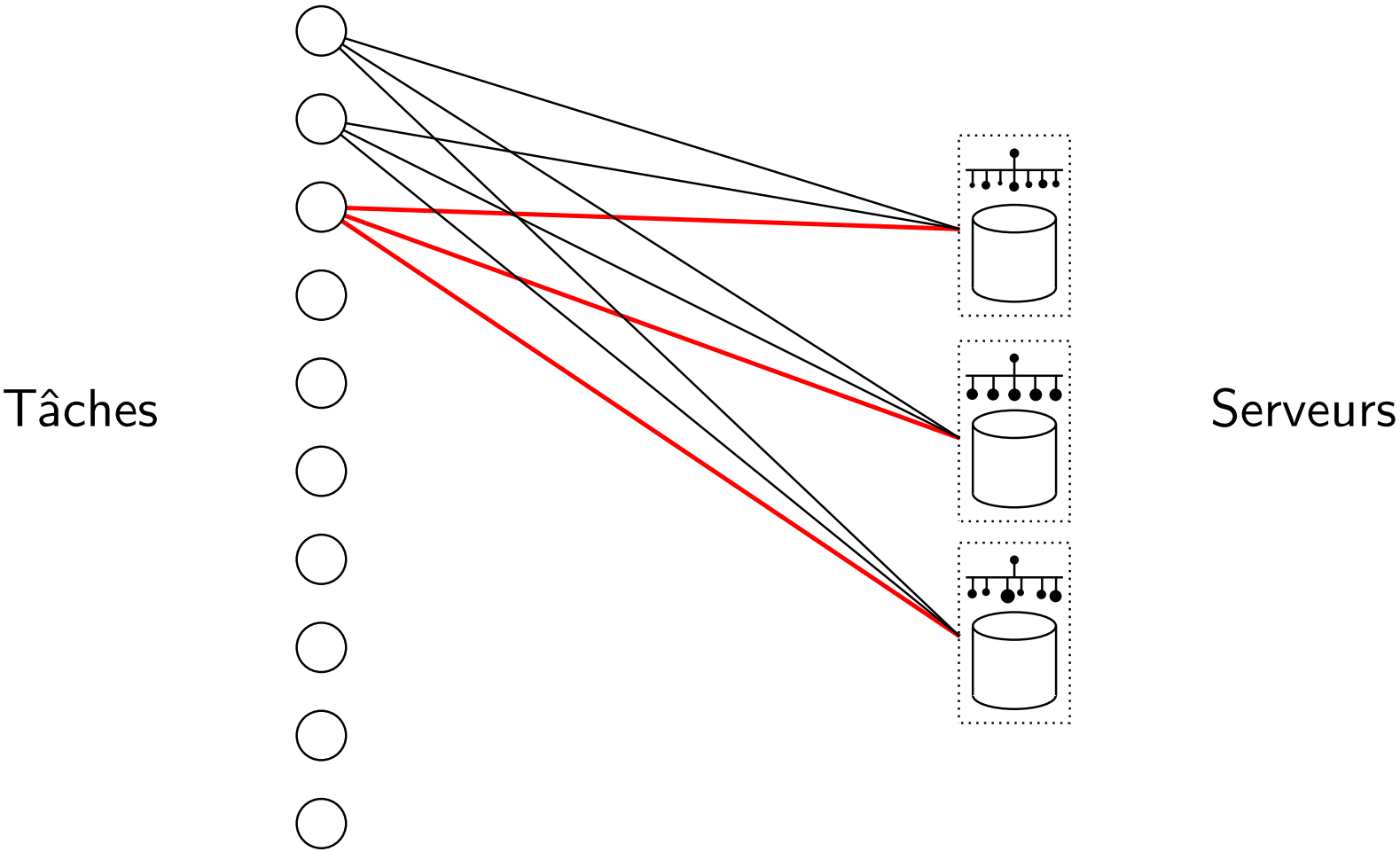
Exemple d'exécution du *min-min*



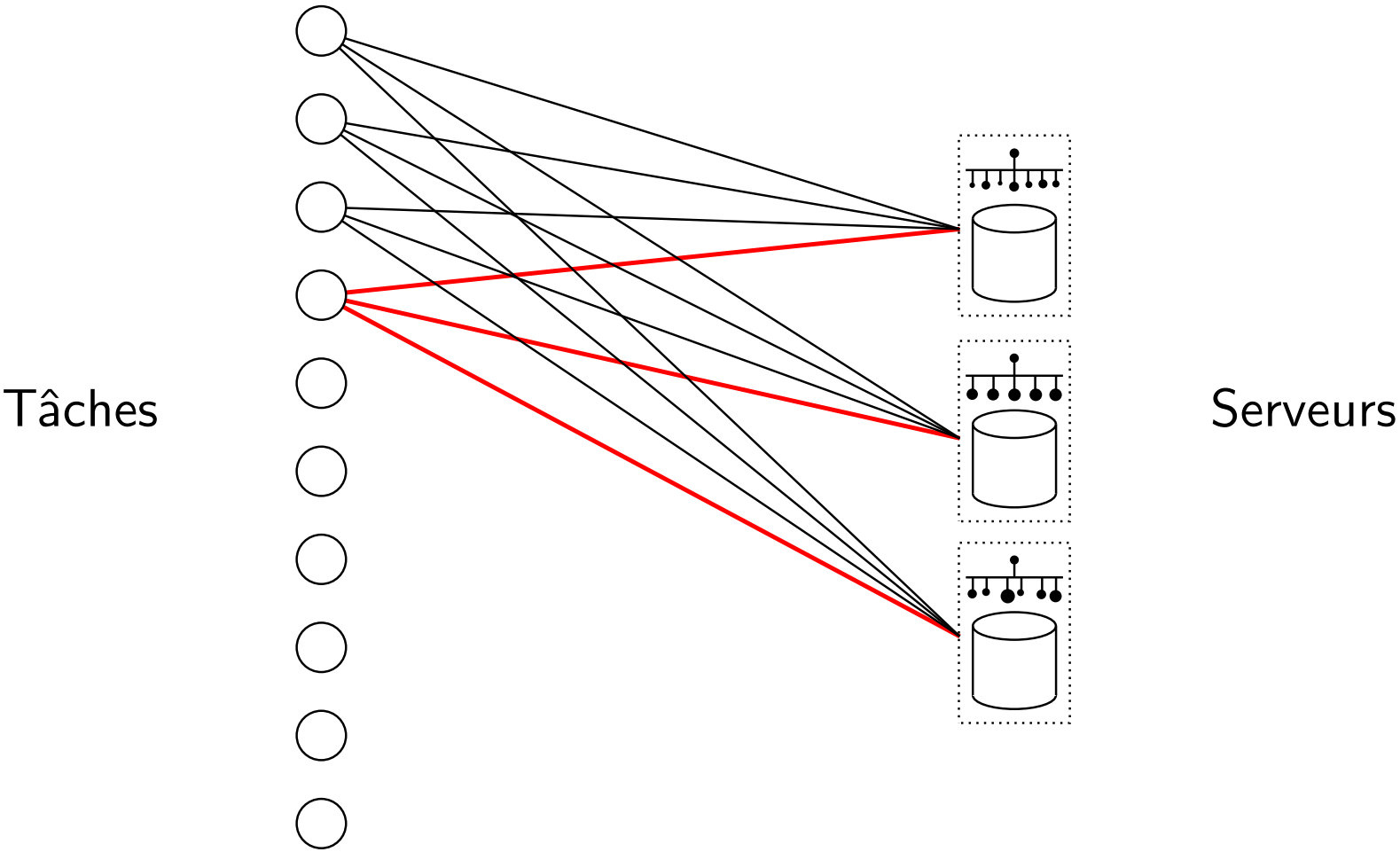
Exemple d'exécution du *min-min*



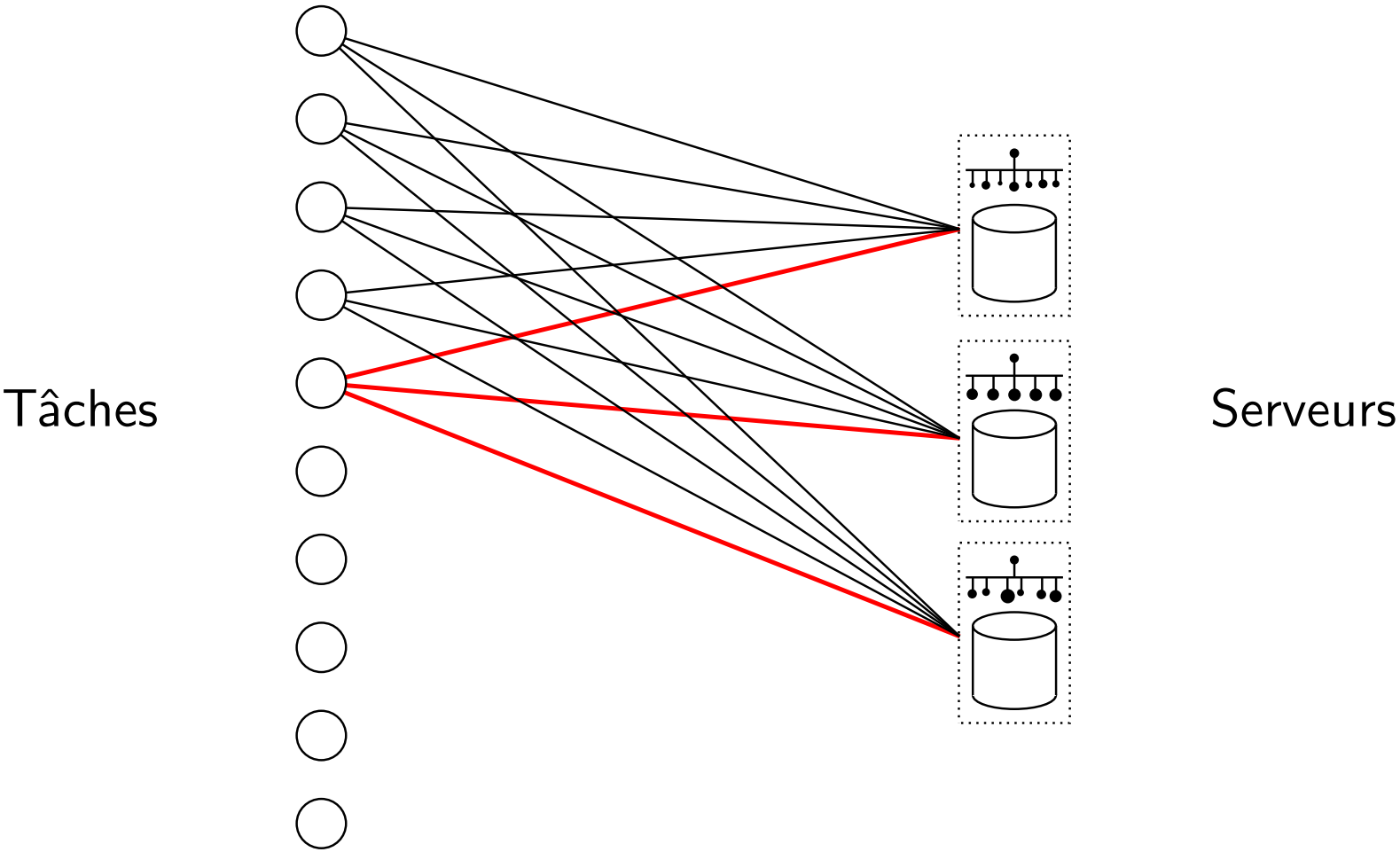
Exemple d'exécution du *min-min*



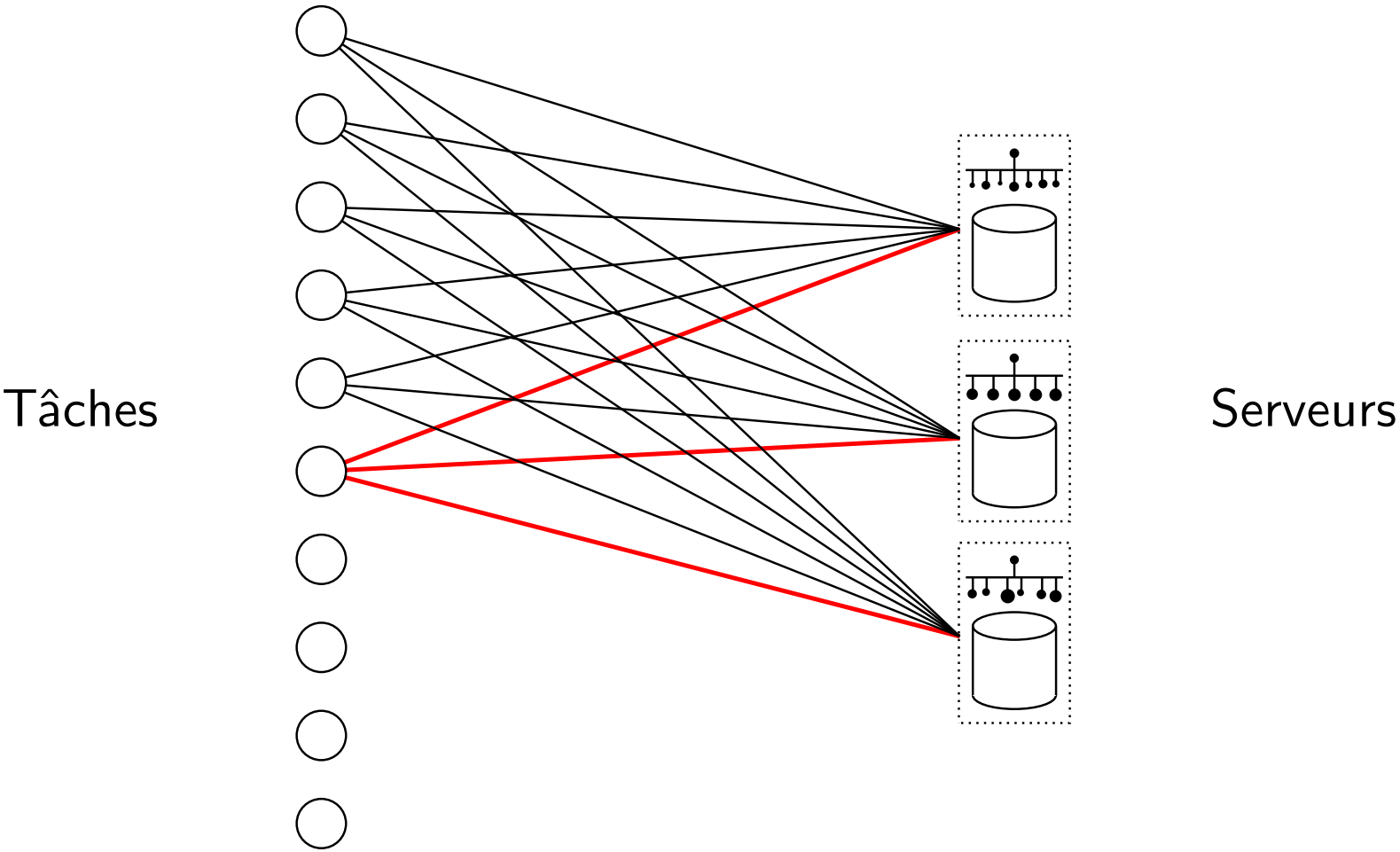
Exemple d'exécution du *min-min*



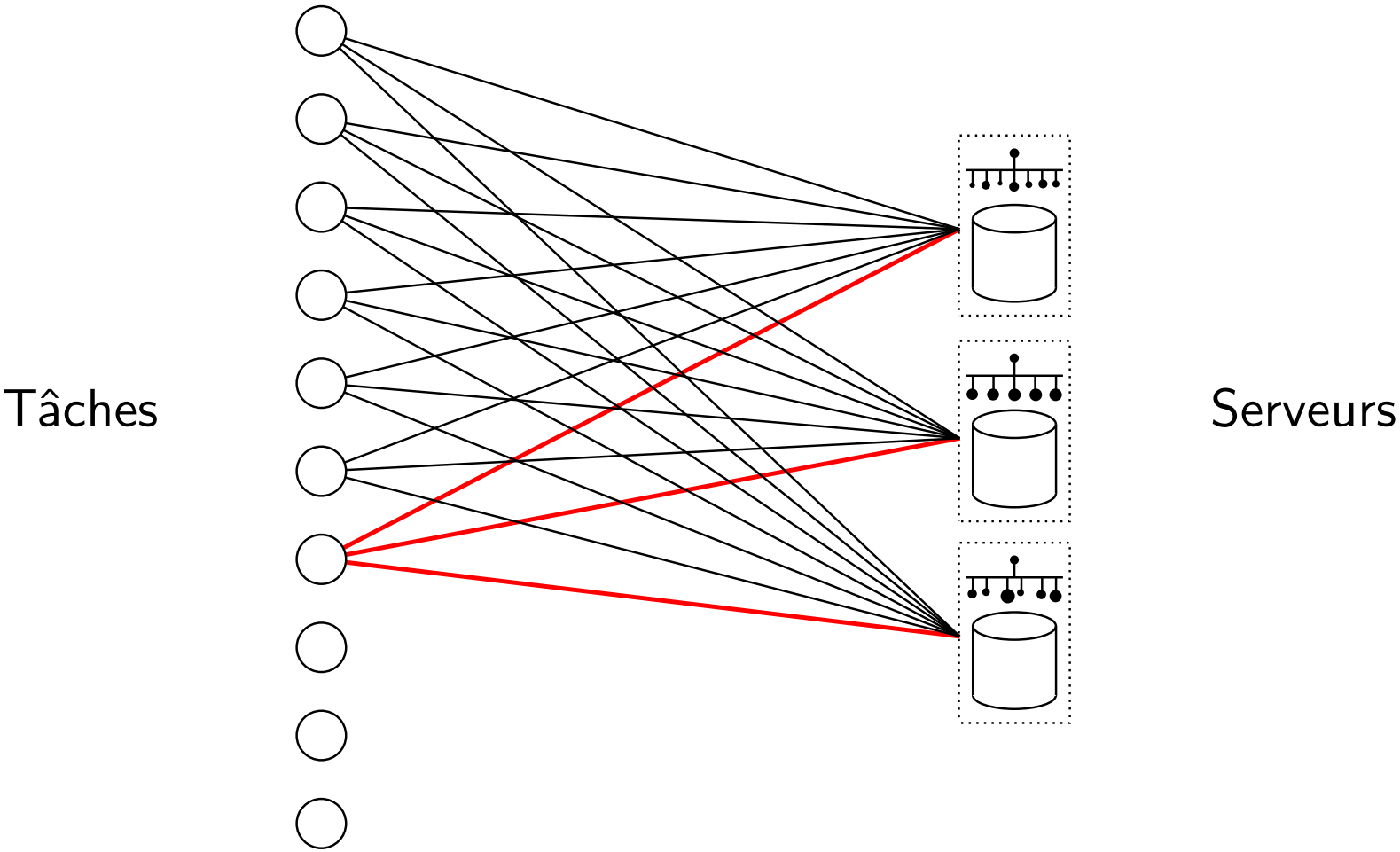
Exemple d'exécution du *min-min*



Exemple d'exécution du *min-min*

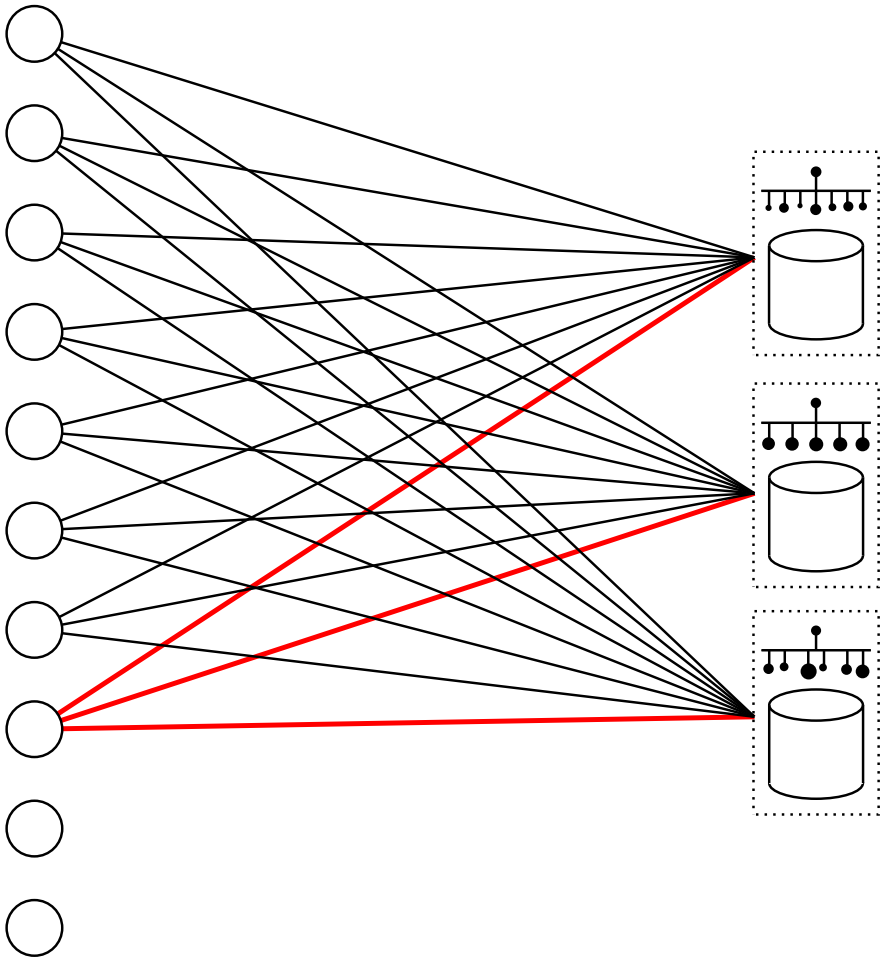


Exemple d'exécution du *min-min*



Exemple d'exécution du *min-min*

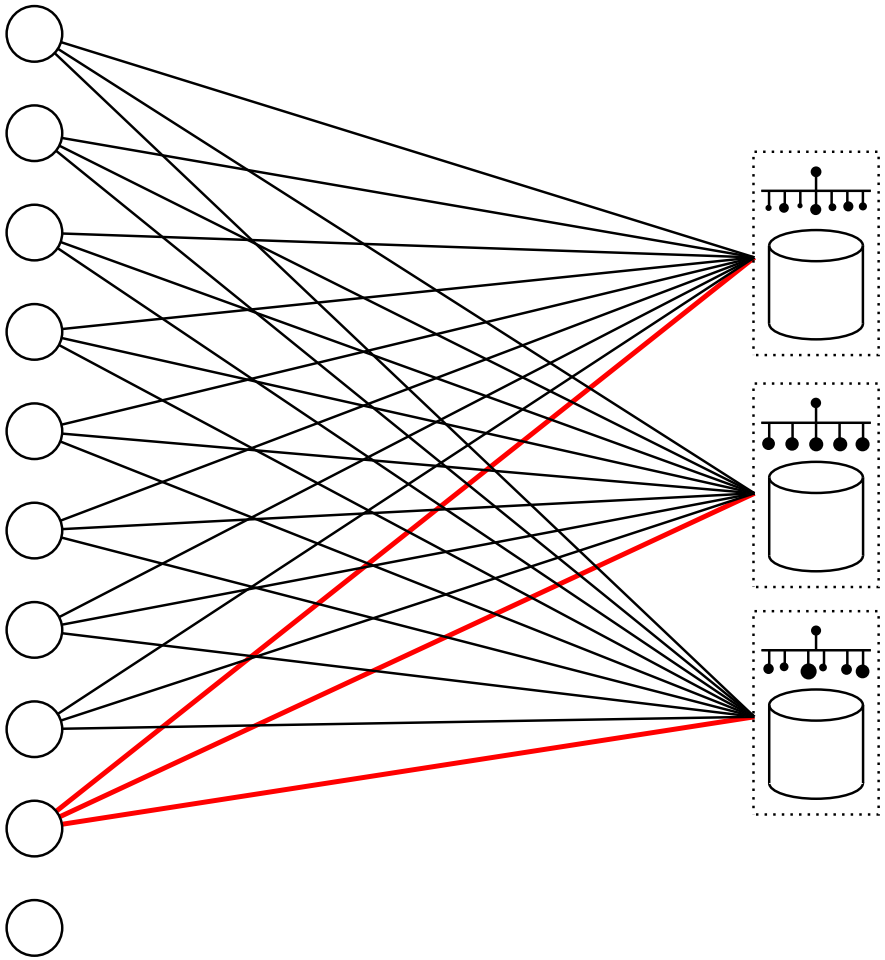
Tâches



Serveurs

Exemple d'exécution du *min-min*

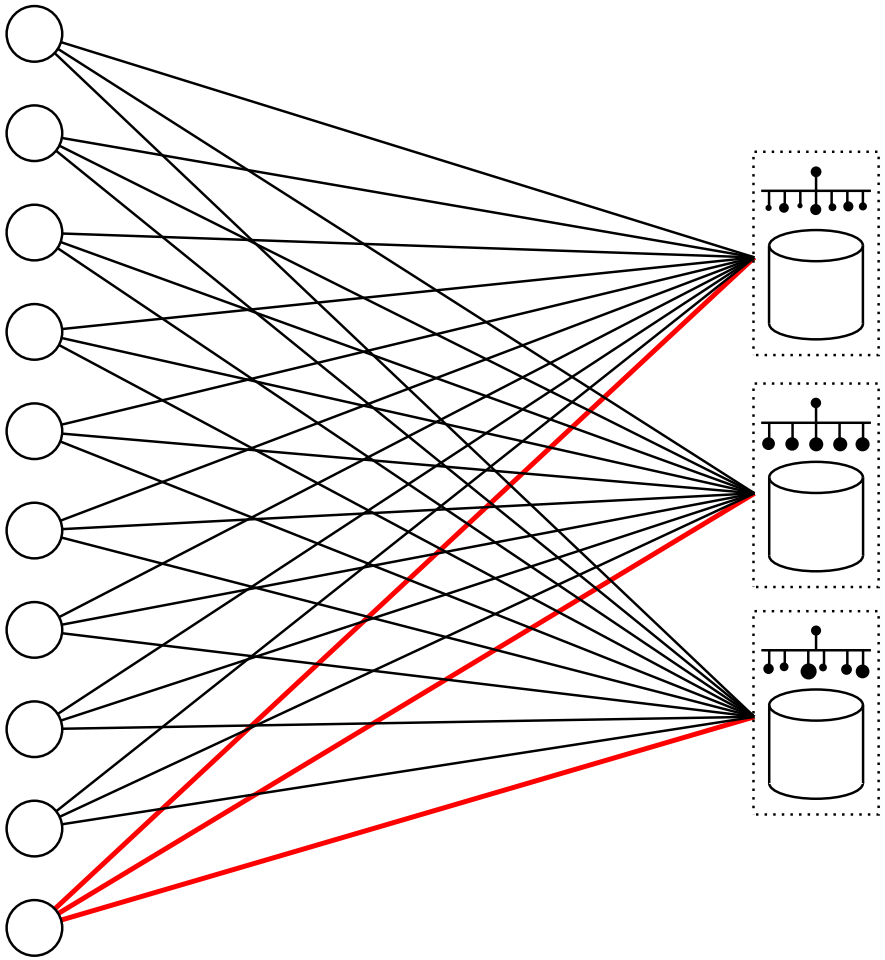
Tâches



Serveurs

Exemple d'exécution du *min-min*

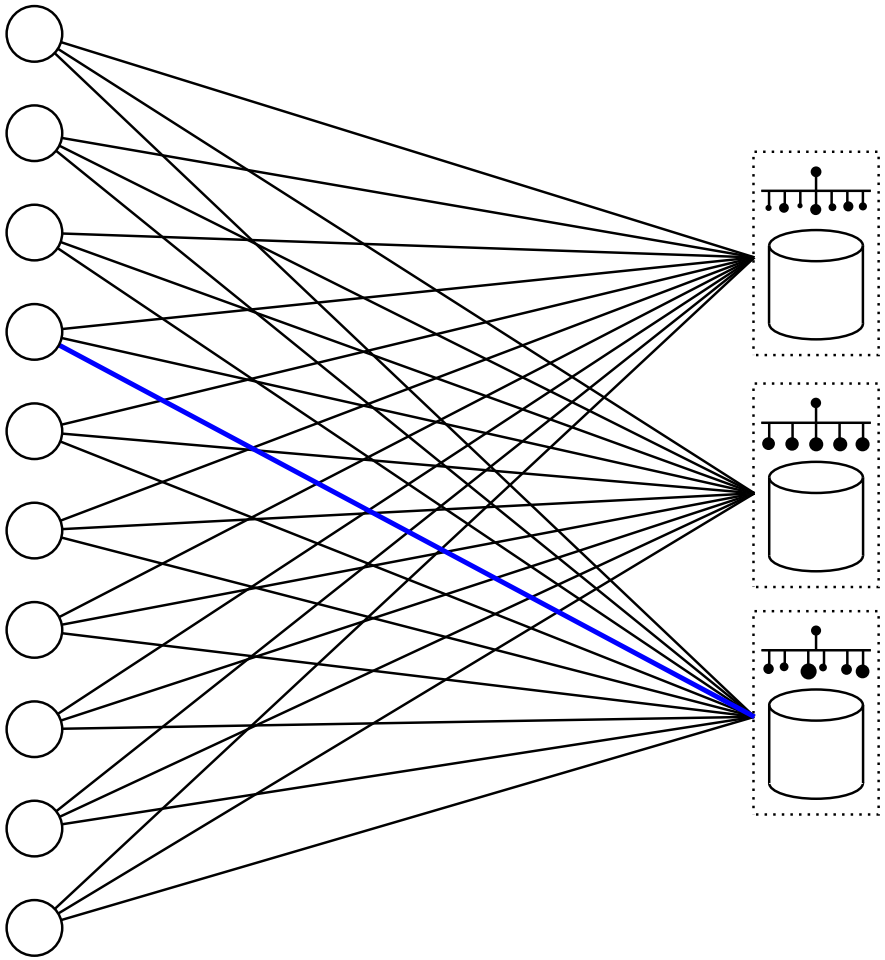
Tâches



Serveurs

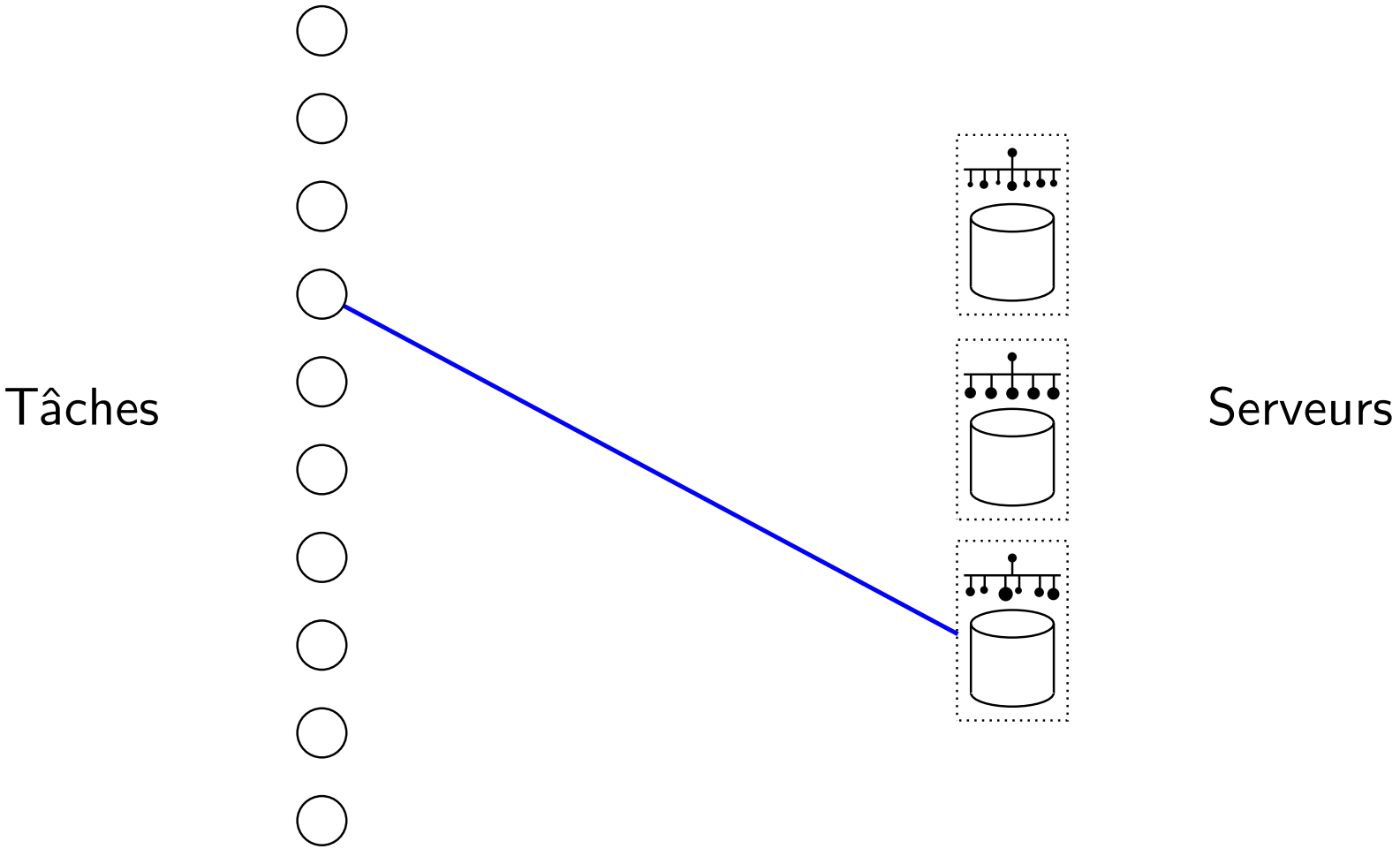
Exemple d'exécution du *min-min*

Tâches



Serveurs

Exemple d'exécution du *min-min*



Adaptation du *min-min* (1/2)

Ordonnancement des communications

- Ordonnancement des communications : NP-complet.
- Heuristiques pour ordonnancer un ensemble de communications de même destination :

  trivial : au plus tôt, *après* la dernière communication

  $O(\Delta\mathcal{P} \cdot \Delta\mathcal{T} \cdot \log \Delta\mathcal{T})$

  insertion : dans le premier intervalle de temps disponible

  $O(\Delta\mathcal{P} \cdot \Delta\mathcal{T} \cdot d)$

- Chaque donnée est transférée depuis le serveur le plus proche.

Adaptation du *min-min* (1/2)

Ordonnancement des communications

- Ordonnancement des communications : NP-complet.
- Heuristiques pour ordonnancer un ensemble de communications de même destination :

  trivial : au plus tôt, *après* la dernière communication
  $O(\Delta\mathcal{P} \cdot \Delta\mathcal{T} \cdot \log \Delta\mathcal{T})$

  insertion : dans le premier intervalle de temps disponible
  $O(\Delta\mathcal{P} \cdot \Delta\mathcal{T} \cdot d)$

- Chaque donnée est transférée depuis le serveur le plus proche.

Adaptation du *min-min* (1/2)

Ordonnancement des communications

- Ordonnancement des communications : NP-complet.
- Heuristiques pour ordonnancer un ensemble de communications de même destination :

  trivial : au plus tôt, *après* la dernière communication
  $O(\Delta\mathcal{P} \cdot \Delta\mathcal{T} \cdot \log \Delta\mathcal{T})$

  insertion : dans le premier intervalle de temps disponible
  $O(\Delta\mathcal{P} \cdot \Delta\mathcal{T} \cdot d)$

- Chaque donnée est transférée depuis le serveur le plus proche.

$\Delta\mathcal{P}$: plus grande distance entre deux serveurs

$\Delta\mathcal{T}$: nombre maximal de données dont dépend une tâche

Adaptation du *min-min* (1/2)

Ordonnancement des communications

- Ordonnancement des communications : NP-complet.
- Heuristiques pour ordonnancer un ensemble de communications de même destination :

  trivial : au plus tôt, *après* la dernière communication
  $O(\Delta\mathcal{P} \cdot \Delta\mathcal{T} \cdot \log \Delta\mathcal{T})$

  insertion : dans le premier intervalle de temps disponible
  $O(\Delta\mathcal{P} \cdot \Delta\mathcal{T} \cdot d)$

- Chaque donnée est transférée depuis le serveur le plus proche.

$\Delta\mathcal{P}$: plus grande distance entre deux serveurs

$\Delta\mathcal{T}$: nombre maximal de données dont dépend une tâche

d : nombre total de données

Adaptation du *min-min* (1/2)

Ordonnancement des communications

- Ordonnancement des communications : NP-complet.
- Heuristiques pour ordonnancer un ensemble de communications de même destination :

  trivial : au plus tôt, *après* la dernière communication
  $O(\Delta\mathcal{P} \cdot \Delta\mathcal{T} \cdot \log \Delta\mathcal{T})$

  insertion : dans le premier intervalle de temps disponible
  $O(\Delta\mathcal{P} \cdot \Delta\mathcal{T} \cdot d)$

- Chaque donnée est transférée depuis le serveur le plus proche.

$\Delta\mathcal{P}$: plus grande distance entre deux serveurs

$\Delta\mathcal{T}$: nombre maximal de données dont dépend une tâche

d : nombre total de données

Adaptation du *min-min* (2/2)

Ordonnancement des tâches

- Les tâches sont d'abord ordonnancées sur les serveurs, en considérant chaque grappe comme une seule ressource de calcul
- L'ordonnancement des tâches au sein de chaque grappe est faite dans un second temps.

Complexité

$$O(s \cdot n^2 \cdot O_c + s^2 \cdot n \cdot |\mathcal{E}| + n \cdot p)$$

n : nombre total de tâches

s : nombre de serveurs

p : nombre maximal de processeurs par grappe

$|\mathcal{E}|$: nombre de relations tâches–données

O_c : coût de l'ordonnancement des communications

Adaptation du *min-min* (2/2)

Ordonnancement des tâches

- Les tâches sont d'abord ordonnancées sur les serveurs, en considérant chaque grappe comme une seule ressource de calcul
- L'ordonnancement des tâches au sein de chaque grappe est faite dans un second temps.

Complexité

$$O(s \cdot n^2 \cdot O_c + s^2 \cdot n \cdot |\mathcal{E}| + n \cdot p)$$

n : nombre total de tâches

s : nombre de serveurs

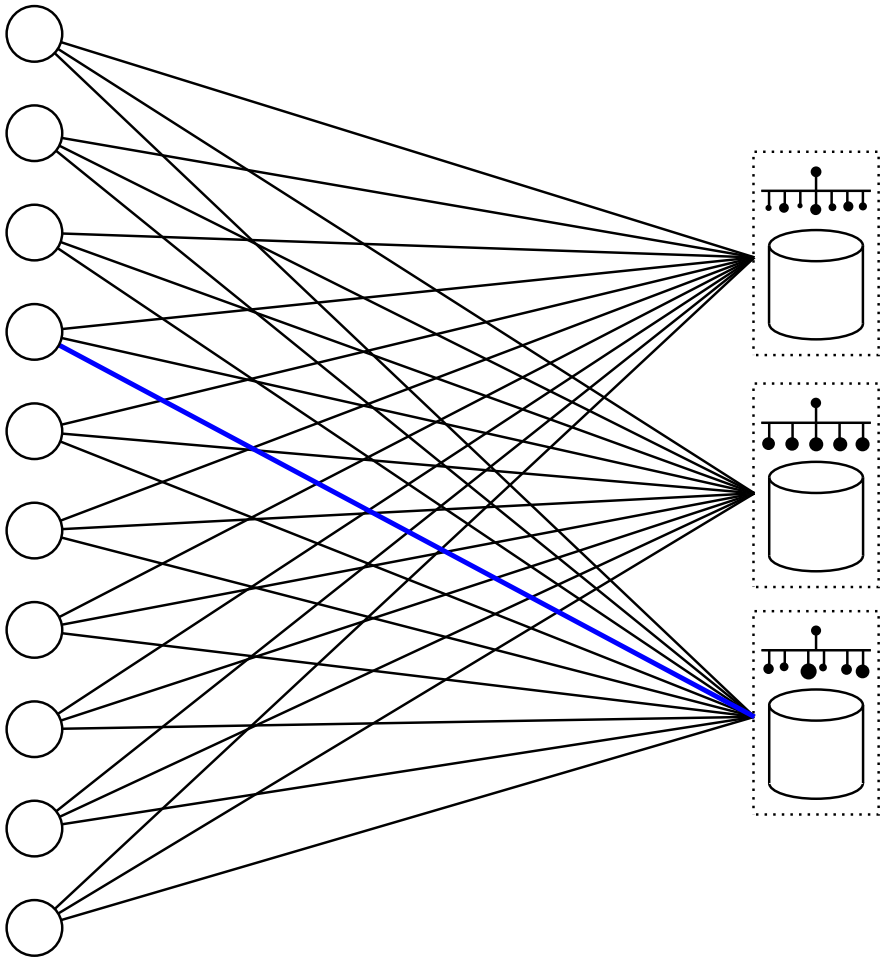
p : nombre maximal de processeurs par grappe

$|\mathcal{E}|$: nombre de relations tâches–données

O_c : coût de l'ordonnancement des communications

Exemple d'exécution du *min-min*

Tâches



Serveurs

Principe des nouvelles heuristiques

Idée

- À chaque étape, au plus une tâche candidate par grappe.

Structure des nouvelles heuristiques

Tant qu'il reste des tâches à ordonnancer, faire

(i) pour chaque grappe C_i, choisir la « meilleure » tâche candidate T_k qui reste à ordonnancer ;

(ii) choisir le « meilleur » couple (T_k, C_i) et ordonnancer T_k sur C_i.

Fonction de coût

- Évaluation de la durée de communication (surestimation) et d'exécution d'une tâche dans le cas où la plate-forme est entièrement disponible.

Principe des nouvelles heuristiques

Idée

- À chaque étape, au plus une tâche candidate par grappe.

Structure des nouvelles heuristiques

Tant qu'il reste des tâches à ordonnancer, faire

(i) pour chaque grappe C_i, choisir la « meilleure » tâche candidate T_k qui reste à ordonnancer ;

(ii) choisir le « meilleur » couple (T_k, C_i) et ordonnancer T_k sur C_i.

Fonction de coût

- Évaluation de la durée de communication (surestimation) et d'exécution d'une tâche dans le cas où la plate-forme est entièrement disponible.

Principe des nouvelles heuristiques

Idée

- À chaque étape, au plus une tâche candidate par grappe.

Structure des nouvelles heuristiques

Tant qu'il reste des tâches à ordonnancer, faire

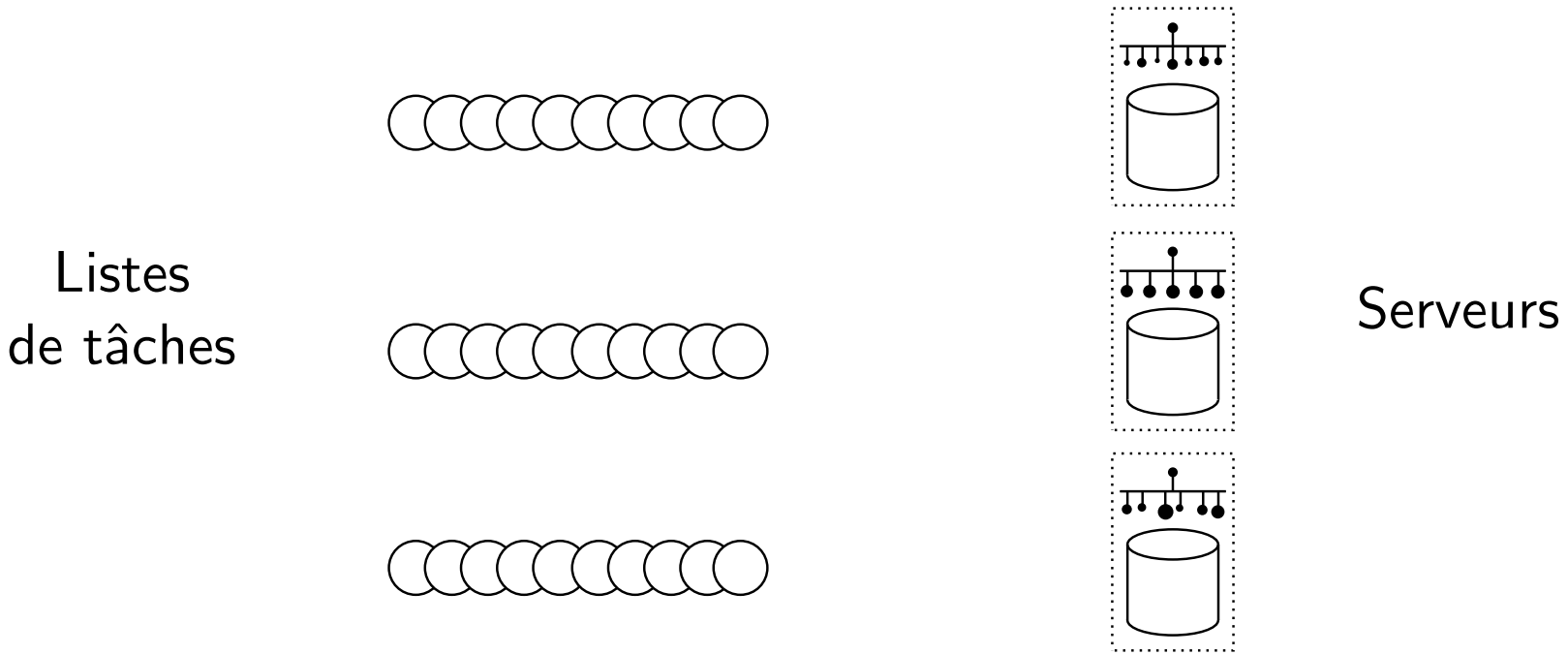
(i) pour chaque grappe C_i, choisir la « meilleure » tâche candidate T_k qui reste à ordonnancer ;

(ii) choisir le « meilleur » couple (T_k, C_i) et ordonnancer T_k sur C_i.

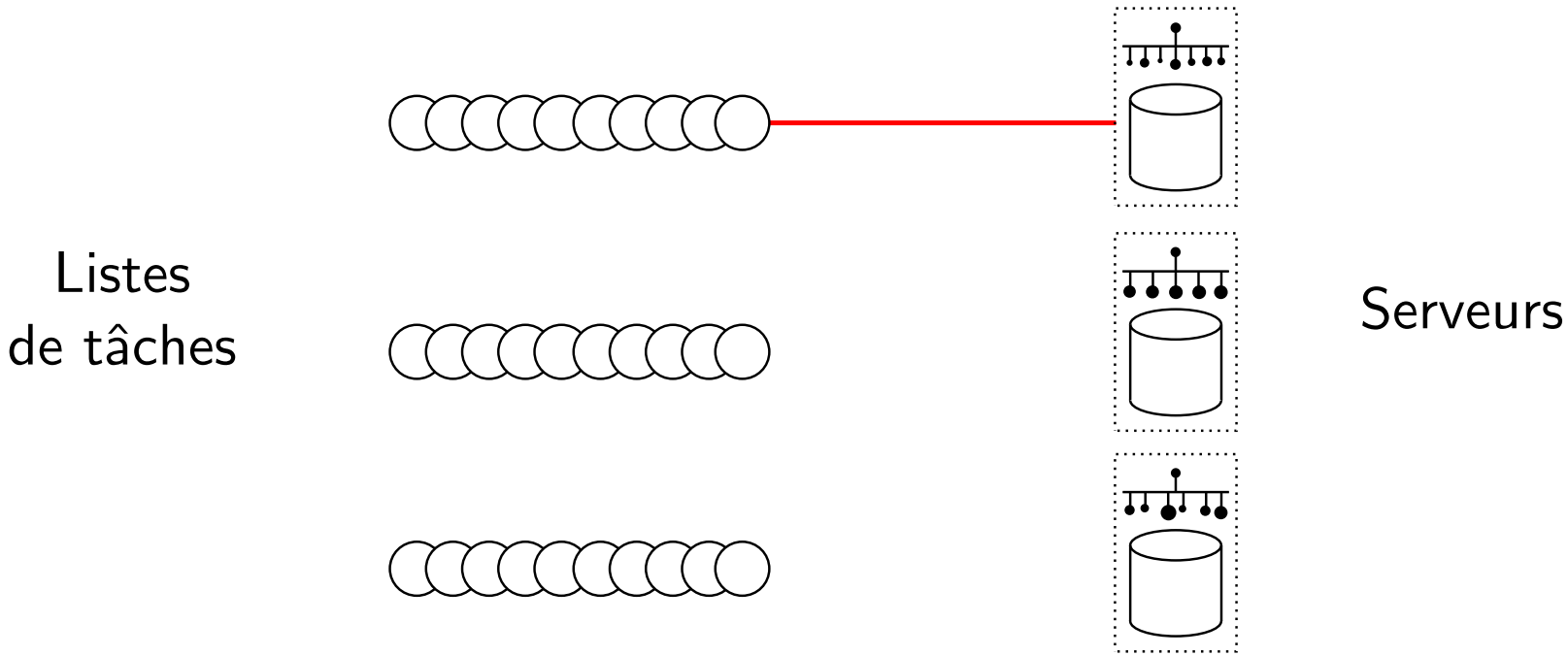
Fonction de coût

- Évaluation de la durée de communication (surestimation) et d'exécution d'une tâche dans le cas où la plate-forme est entièrement disponible.

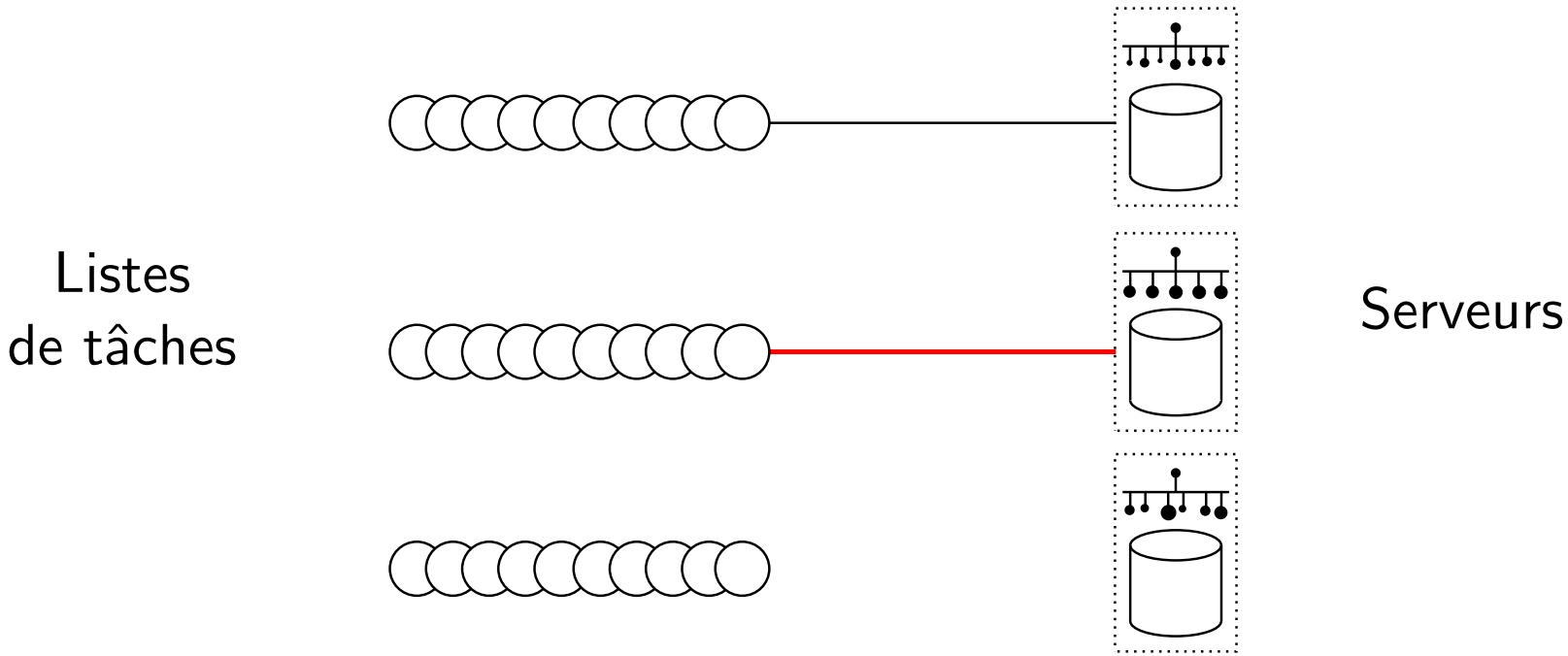
Exemple d'exécution des nouvelles heuristiques



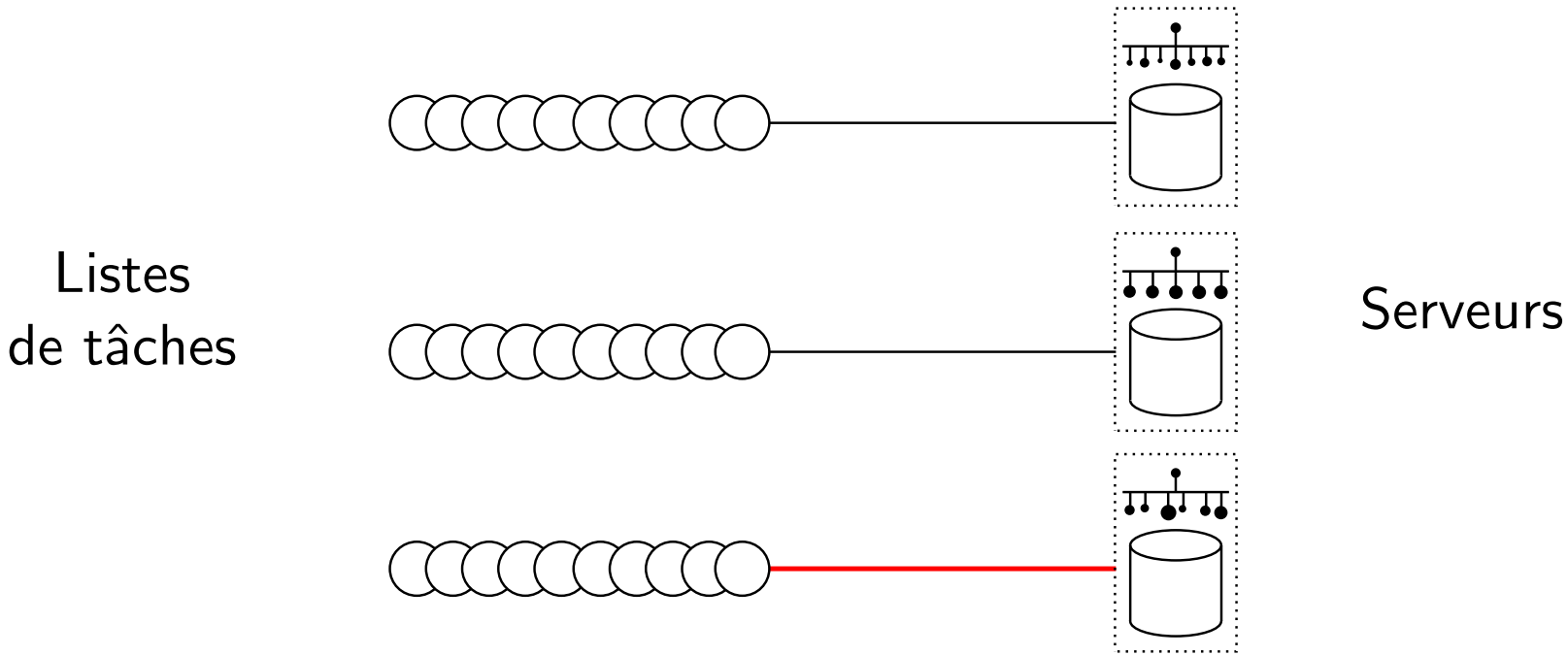
Exemple d'exécution des nouvelles heuristiques



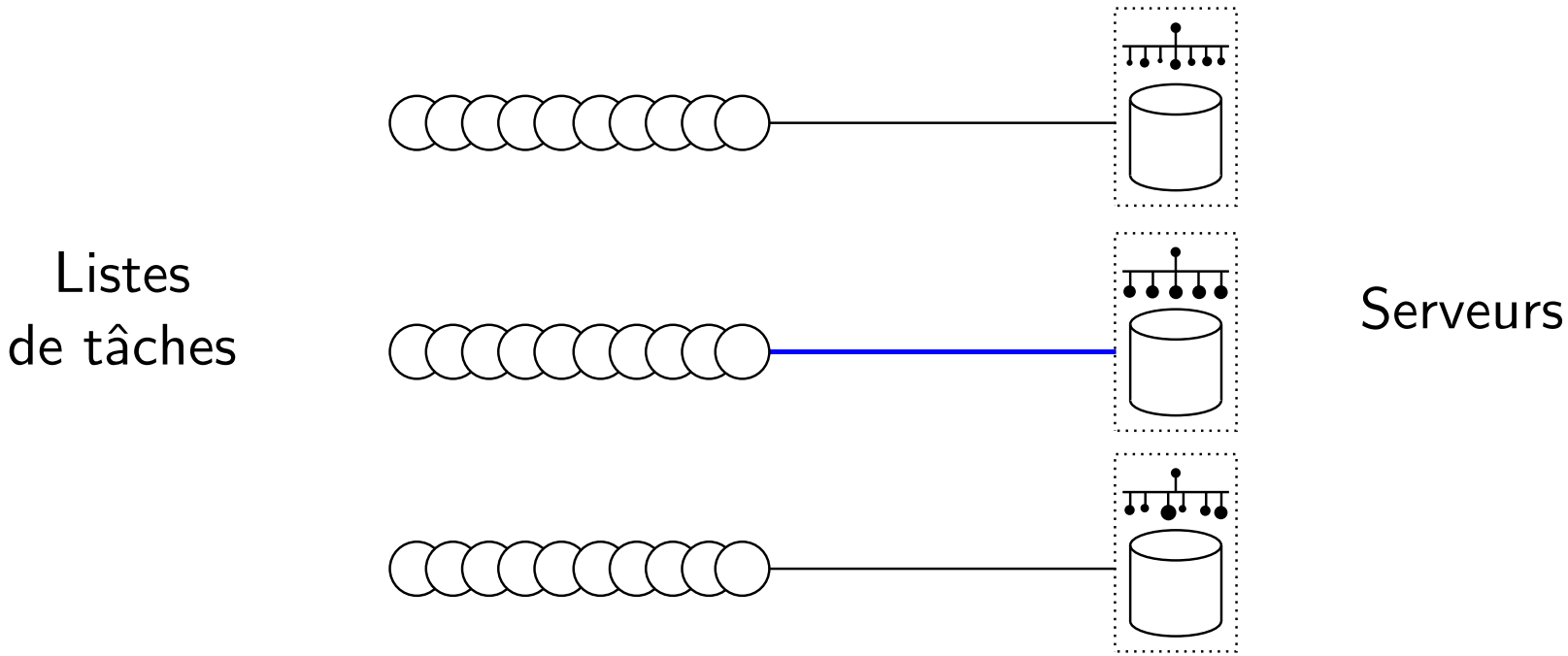
Exemple d'exécution des nouvelles heuristiques



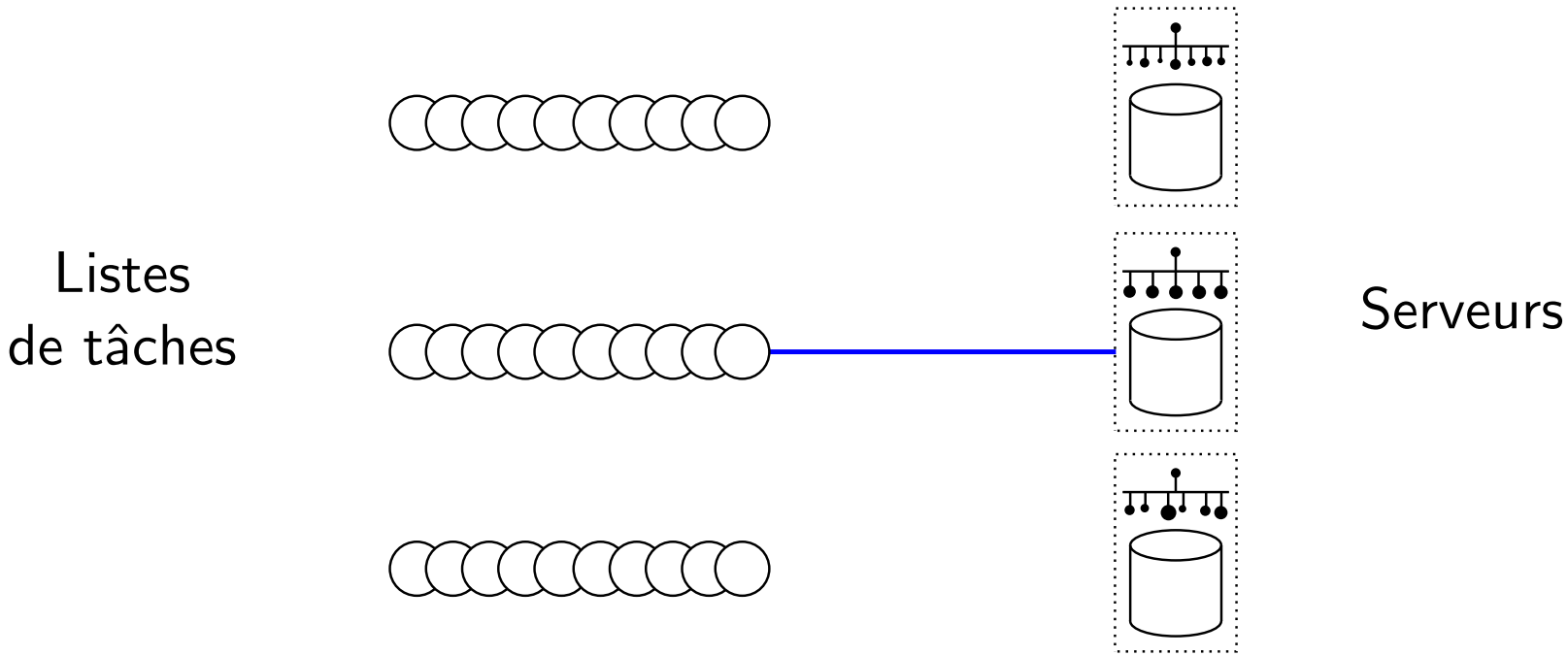
Exemple d'exécution des nouvelles heuristiques



Exemple d'exécution des nouvelles heuristiques



Exemple d'exécution des nouvelles heuristiques



Heuristiques statiques

Principe de l'heuristique

- Candidat local pour la grappe C_i :
  tâche non ordonnancée de plus petit coût
- Nous prenons le couple (T_k, C_i) de plus petit coût
- Nous ordonnançons les communications nécessaires pour exécuter T_k sur C_i
- Nous ordonnançons T_k sur C_i

Complexité

$$O(n \cdot (\log n + O_c) + s^2 \cdot \Delta\mathcal{P} + s \cdot |\mathcal{E}| + n \cdot p))$$

Variantes

Deux améliorations :

1. Variante *readiness* : si une tâche est *prête* sur un serveur (pas de communication), elle est sélectionnée au lieu de la tâche de plus petit coût.
   Surcoût négligeable : incluse par défaut dans les heuristiques.
2. Variante *mct* : les différentes tâches candidates sont mises en concurrence par une évaluation de leur MCT.
   $O(s \cdot n \cdot (\log n + O_c) + s^2 \cdot (\Delta\mathcal{P} + |\mathcal{E}|) + n \cdot p)$

Variantes

Deux améliorations :

1. Variante *readiness* : si une tâche est *prête* sur un serveur (pas de communication), elle est sélectionnée au lieu de la tâche de plus petit coût.
   Surcoût négligeable : incluse par défaut dans les heuristiques.
2. Variante *mct* : les différentes tâches candidates sont mises en concurrence par une évaluation de leur MCT.
   $O(s \cdot n \cdot (\log n + O_c) + s^2 \cdot (\Delta\mathcal{P} + |\mathcal{E}|) + n \cdot p)$

Heuristiques dynamiques

- L'estimation du *coût* devient de moins en moins pertinente au fur et à mesure que l'exécution progresse.
- Nous définissons un *coût dynamique* dont la valeur est mise à jour à chaque fois qu'une donnée D est répliquée, mais seulement pour les tâches dépendant de D.
- Une structure de données de type tas permet de déterminer l'élément minimum avec un coût raisonnable.

Toutes les variantes réunies : 81 heuristiques.

Heuristiques dynamiques

- L'estimation du *coût* devient de moins en moins pertinente au fur et à mesure que l'exécution progresse.
- Nous définissons un *coût dynamique* dont la valeur est mise à jour à chaque fois qu'une donnée D est répliquée, mais seulement pour les tâches dépendant de D.
- Une structure de données de type tas permet de déterminer l'élément minimum avec un coût raisonnable.

Toutes les variantes réunies : 81 heuristiques.

Simulations
Plates-formes simulées

- 7, 50 ou 100 serveurs
- Réseau d'interconnexion : clique, anneau ou arbre aléatoire
- Chaque grappe composée de 8, 16 ou 32 processeurs
- Vitesses des processeurs tirées aléatoirement d'un ensemble de valeurs réelles
- Bandes passantes des liens de communication tirées aléatoirement d'un ensemble de valeurs réelles

Ratio des coûts de communication et de calcul

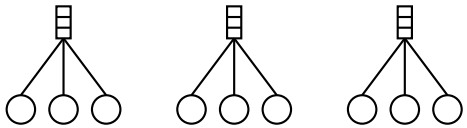
Normalisation des moyennes des caractéristiques des processeurs et des communications, de manière à modéliser trois principaux types de problèmes :

- intensif en calculs (ratio $= 0{,}1$) ;
- intensif en communications (ratio $= 10$) ;
- intermédiaire (ratio $= 1$).

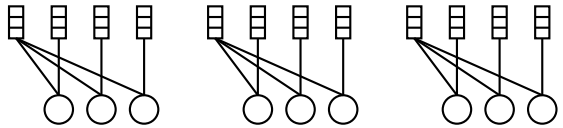
Simulations

Graphes d'application (avec 7 serveurs)

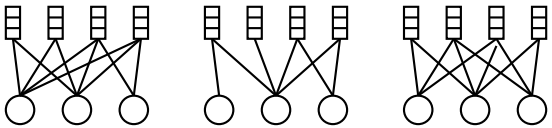
- Tailles des données et des tâches tirées aléatoirement entre 0,5 et 5.



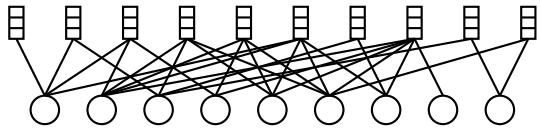
étoile (70 × 22 tâches/donnée)



deux-un (2 données/tâche)



partitionné (20 cc., 1–10 données/tâche)



aléatoire (1–50 données/tâche)

- Chaque graphe : 1 500 tâches et 1 750 données (sauf les graphes en étoile : 70 données).
- Sommes des tailles des tâches et des données normalisées, de manière à respecter le ratio des coûts de communication et de calcul.

Simulations
Évaluation

- Deux caractéristiques par heuristique :

  performance : temps de fin simulé pour l'ordonnancement ;

  coût : temps nécessaire à la construction de l'ordonnancement.
- Ce sont des valeurs absolues qui ne sont pas comparables entre les différentes configurations.

- Utilisation, pour chaque configuration, de performances relatives :

$$\text{performance relative}(H) = \frac{\text{performance}(H)}{\min_{h \in \{\text{heuristiques}\}} \text{performance}(h)}.$$

  La moyenne est ensuite calculée sur toutes les configurations.
- Les coûts relatifs sont définis de la même manière.

Résultats

	7 serveurs		50 serveurs		100 serveurs	
Heuristique	Performance	Coût	Performance	Coût	Performance	Coût
sufferage +insert	1,06 (±10%)	21 984 (±99%)				
min-min +insert	1,07 (±7%)	18 002 (±104%)				
dynamic +mct+insert	1,10 (±9%)	19 (±72%)	1,05 (±9%)	873 (±117%)		
randomtask +mct+insert	1,10 (±10%)	14 (±67%)	1,05 (±8%)	1 356 (±204%)		
static +mct+insert	1,19 (±13%)	13 (±65%)	1,17 (±14%)	843 (±184%)		
dynamic +mct	1,34 (±17%)	14 (±79%)	1,62 (±34%)	299 (±75%)	1,04 (±8%)	568 (±80%)
randomtask +mct	1,37 (±22%)	7 (±80%)	1,84 (±43%)	11 (±48%)	1,16 (±16%)	18 (±67%)
static+mct	1,48 (±24%)	7 (±92%)	1,88 (±40%)	10 (±53%)	1,19 (±21%)	17 (±63%)
static	2,19 (±33%)	3 (±104%)	4,37 (±61%)	1 (±41%)		
randomtask	130 (±318%)	2 (±92%)	1 377 (±419%)	1 (±12%)		

7 serveurs : moyennes sur 48 000 configurations

50 serveurs : moyennes sur 2 400 configurations

100 serveurs : moyennes sur 1 056 configurations

Conclusion

- Ordonnancement d'un grand ensemble de tâches partageant des données.
- Plates-formes hétérogènes entièrement distribuées.
- Étude de la complexité théorique du problème.
- Extension de l'heuristique *min-min*.
- Nouvelles heuristiques aussi bonnes mais d'un ordre de grandeur moins coûteuses que les heuristiques de référence.

Conclusion et perspectives

Conclusion

- Étude de stratégies d'ordonnancement et d'équilibrage de charge pour plates-formes hétérogènes distribuées.
- Tâches indépendantes, partage des données.
- Généralisation progressive du problème.
- Étude de la complexité théorique.
- Algorithmes pour les cas simples, validation expérimentale.
- Nouvelles heuristiques pour les cas plus complexes.

Perspectives

- Complexité : il reste des problèmes ouverts.
- Une approche plus globale, par partitionnement du graphe d'application.
- Modèle de communication : multi-port, partage de bande passante.
- Robustesse : aux erreurs de prédiction, aux perturbations de la plate-forme.

Table des matières