Parallélisme sur grilles avec P2P-MPI

Présentation Master UdS

Stéphane Genaud

LSIIT-ICPS, Université de Strasbourg
http://www.p2pmpi.org/

April 10, 2011

Grilles: difficultés principales

- machines appartenant à des propriétaires différents
- machines distribuées géographiquement, souvent à large échelle

Conséquences:

- connexions réseaux hétérogènes
- puissances de calcul hétérogènes
- hétérogénéité des systèmes et logiciels
- pannes fréquentes
- système d'information pas maîtrisé

Caractéristiques de P2P-MPI

Un environnement offrant: une implémentation MPJ + un intergiciel de gestion P2P des ressources

- Installation et développement très simples (1 jar)
- Décentralisé : environnement pair-à-pair
- Exécute des bytecodes Java
- Construit de manière automatique une plate-forme à chaque exécution
- Transfert automatique des fichiers (bytecode et input)
- Tolérant aux pannes (détection pannes & réplication processus)

Caractéristiques de P2P-MPI

Un environnement offrant: une implémentation MPJ + un intergiciel de gestion P2P des ressources

- Installation et développement très simples (1 jar)
- Décentralisé : environnement pair-à-pair
- Exécute des bytecodes Java
- Construit de manière automatique une plate-forme à chaque exécution
- Transfert automatique des fichiers (bytecode et input)
- Tolérant aux pannes (détection pannes & réplication processus)

Caractéristiques de P2P-MPI

Un environnement offrant: une implémentation MPJ + un intergiciel de gestion P2P des ressources

- Installation et développement très simples (1 jar)
- Décentralisé : environnement pair-à-pair
- Exécute des bytecodes Java
- Construit de manière automatique une plate-forme à chaque exécution
- Transfert automatique des fichiers (bytecode et input)
- Tolérant aux pannes (détection pannes & réplication processus)

Caractéristiques de P2P-MPI

Un environnement offrant: une implémentation MPJ + un intergiciel de gestion P2P des ressources

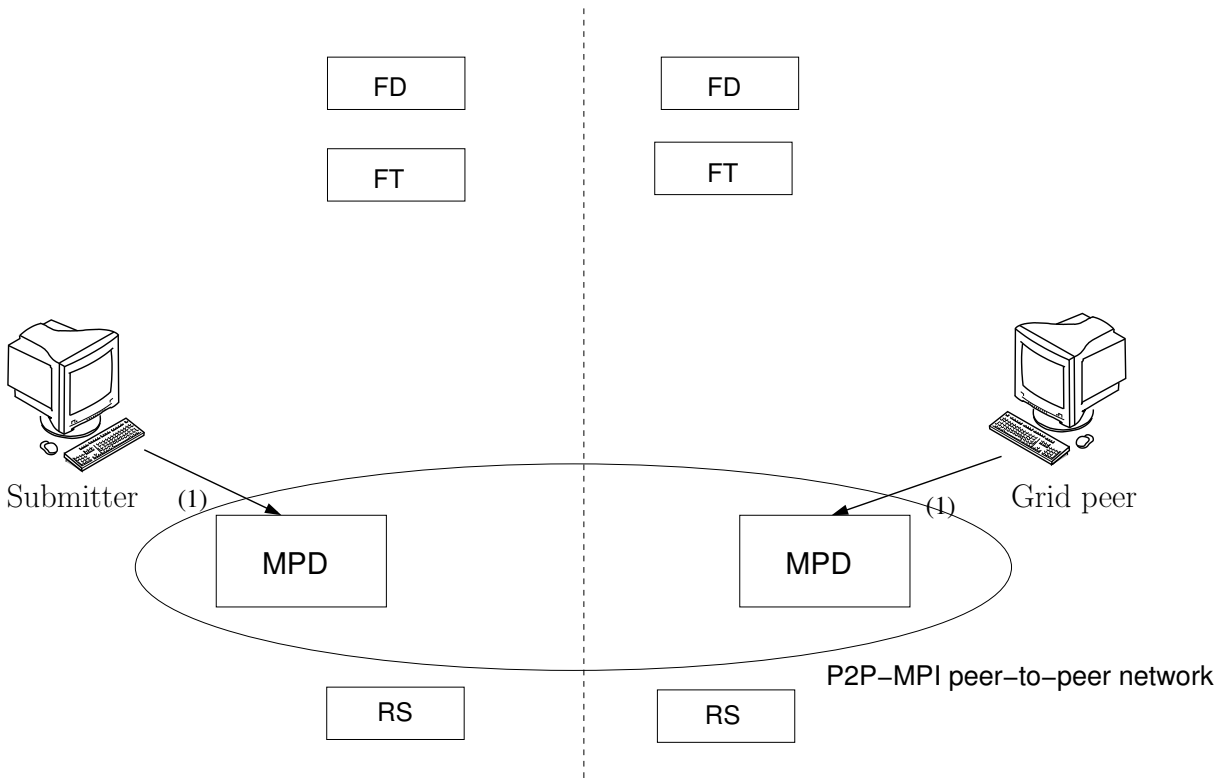
- Installation et développement très simples (1 jar)
- Décentralisé : environnement pair-à-pair
- Exécute des bytecodes Java
- Construit de manière automatique une plate-forme à chaque exécution
- Transfert automatique des fichiers (bytecode et input)
- Tolérant aux pannes (détection pannes & réplication processus)

Principe : l'utilisateur offre sa machine, et profite de celle des autres.

Activité de l'utilisateur

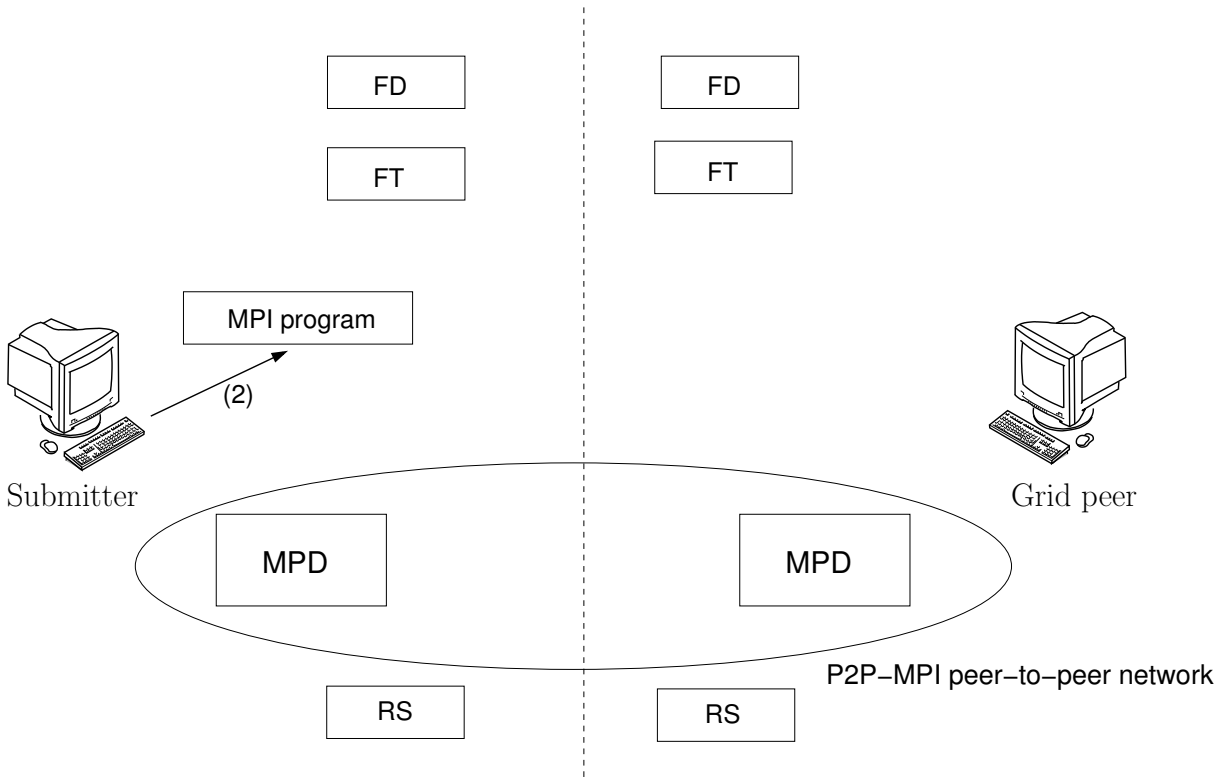
- configure la capacité CPU offerte
- configure le supernode
- `mpiboot` enregistre la participation dans le réseau P2P
- développe un programme parallèle MPJ
- `p2pmpirun` lance le programme en utilisant les CPU des autres
- `mpihalt` retire sa machine du réseau P2P

Scenario démarrage application



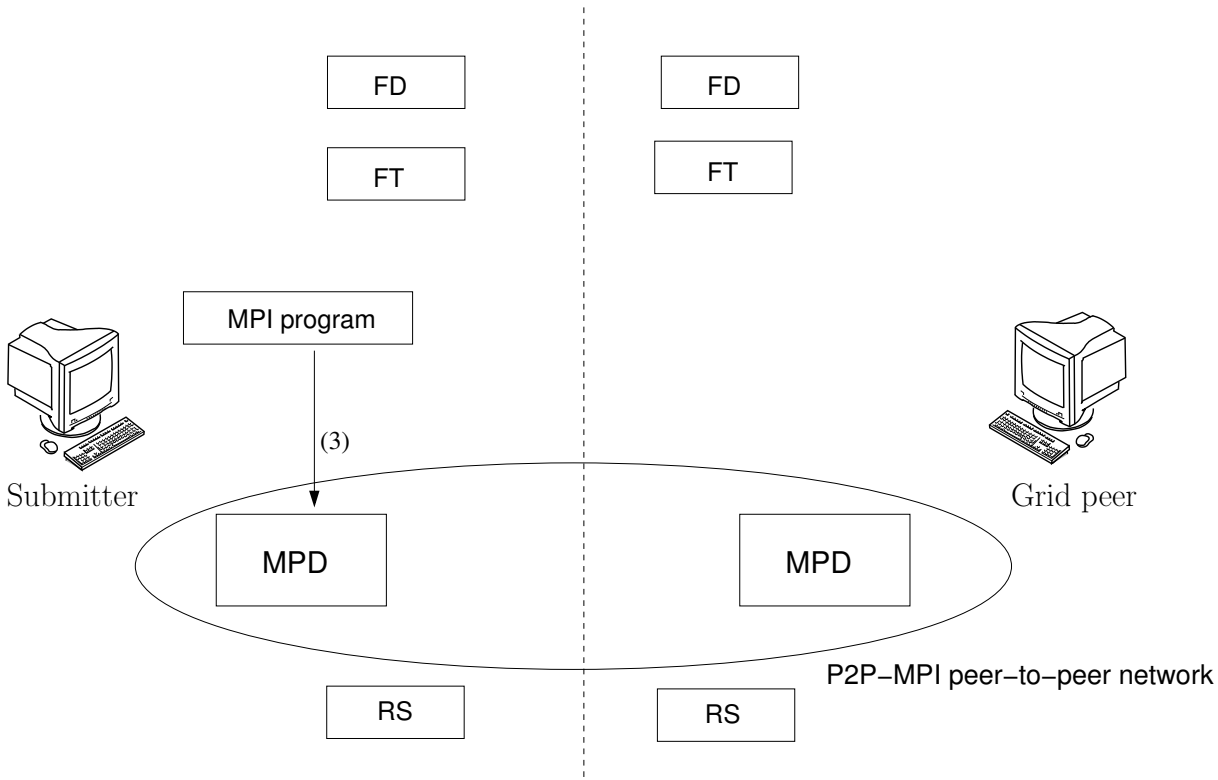
Booting up: `mpiboot` starts `MPD, FD, FT, RS.`

Scenario démarrage application



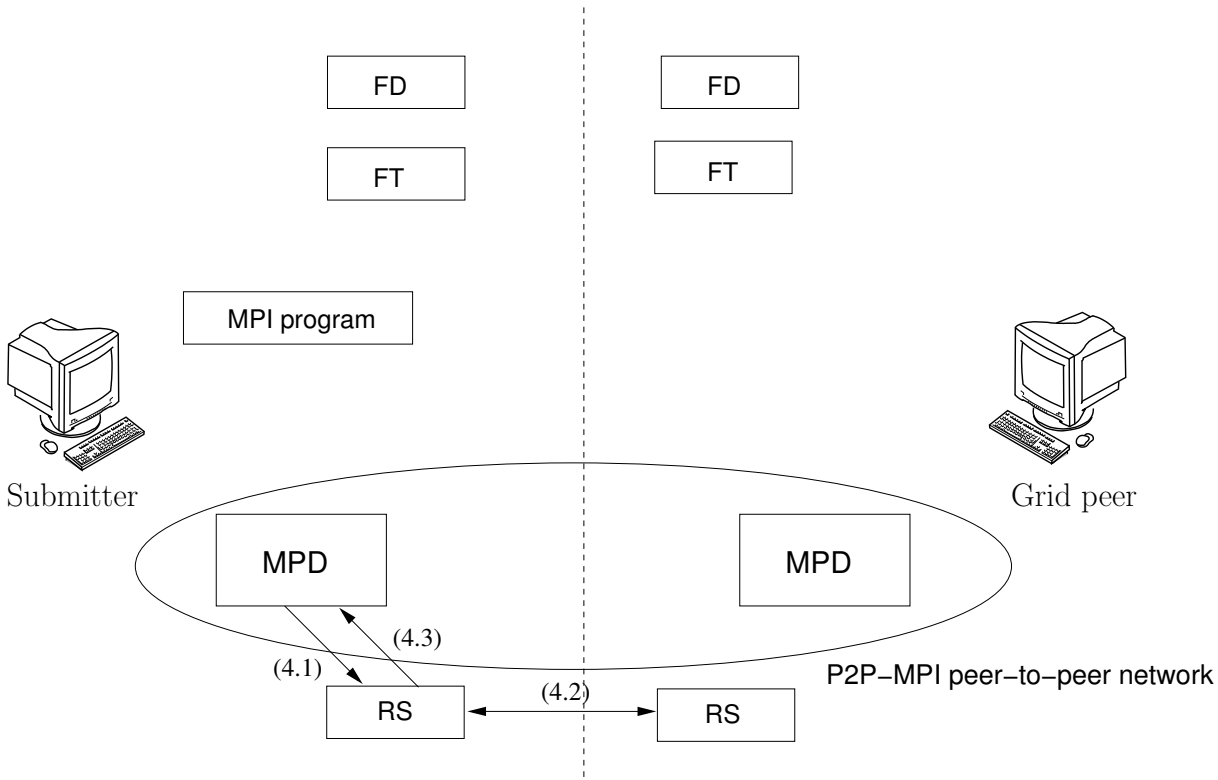
Job submission: `p2pmpirun -n` *n* `-r` *r* `-a` *alloc* `prog`.

Scenario démarrage application



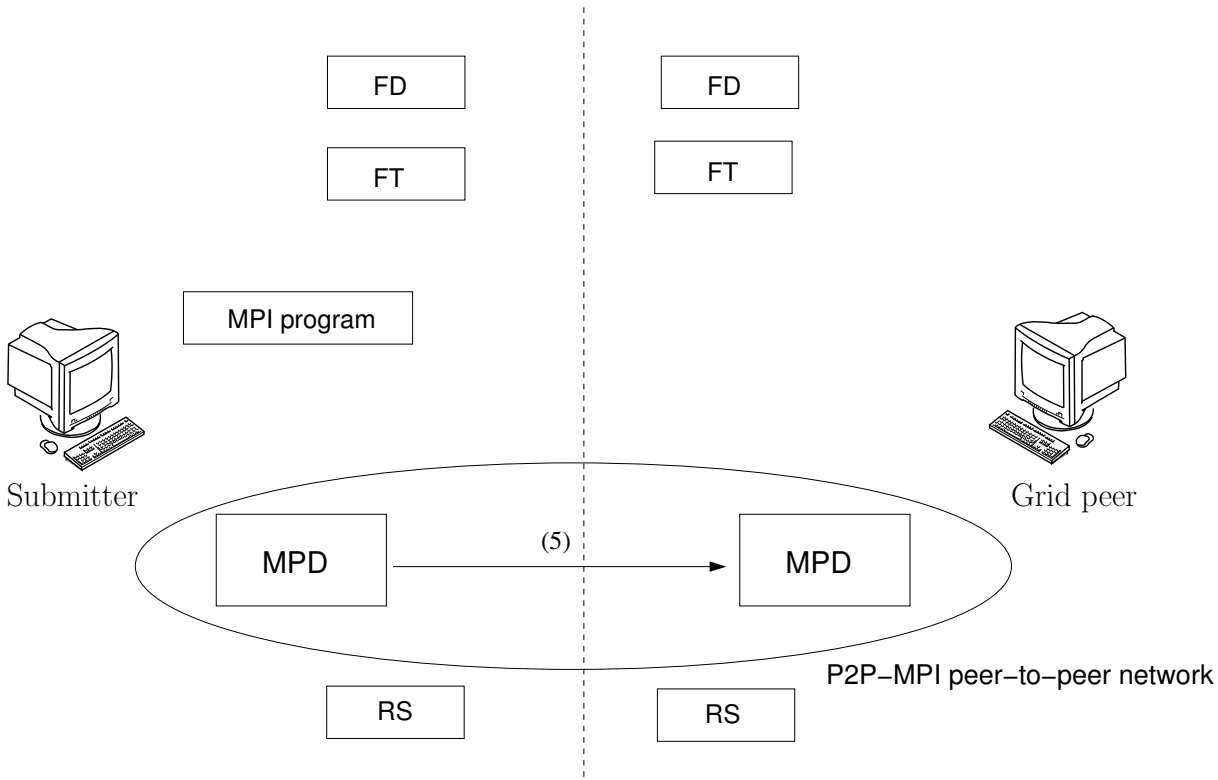
Requesting peers: Application asks `MPD` to discover resources for executing $n \times r$ MPI processes.

Scenario démarrage application



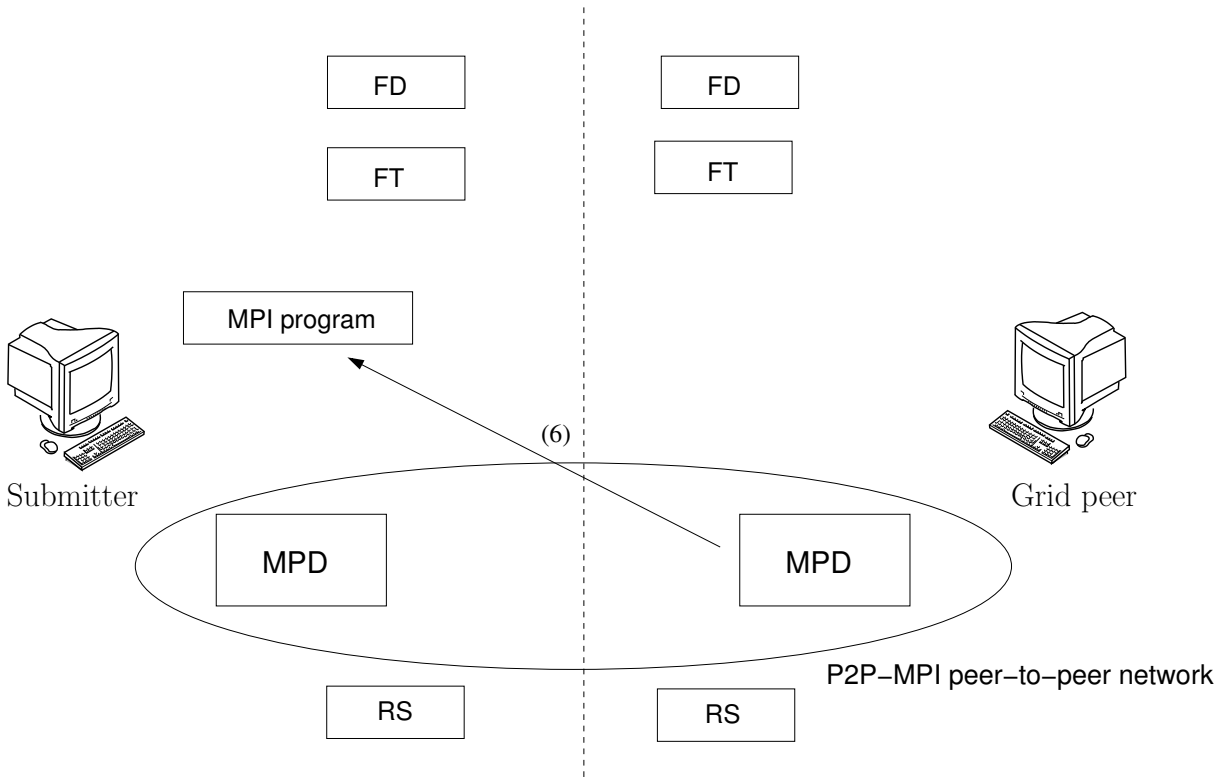
Discovery and Reservation: `MPD` requests `RS` to reserve peer.

Scenario démarrage application



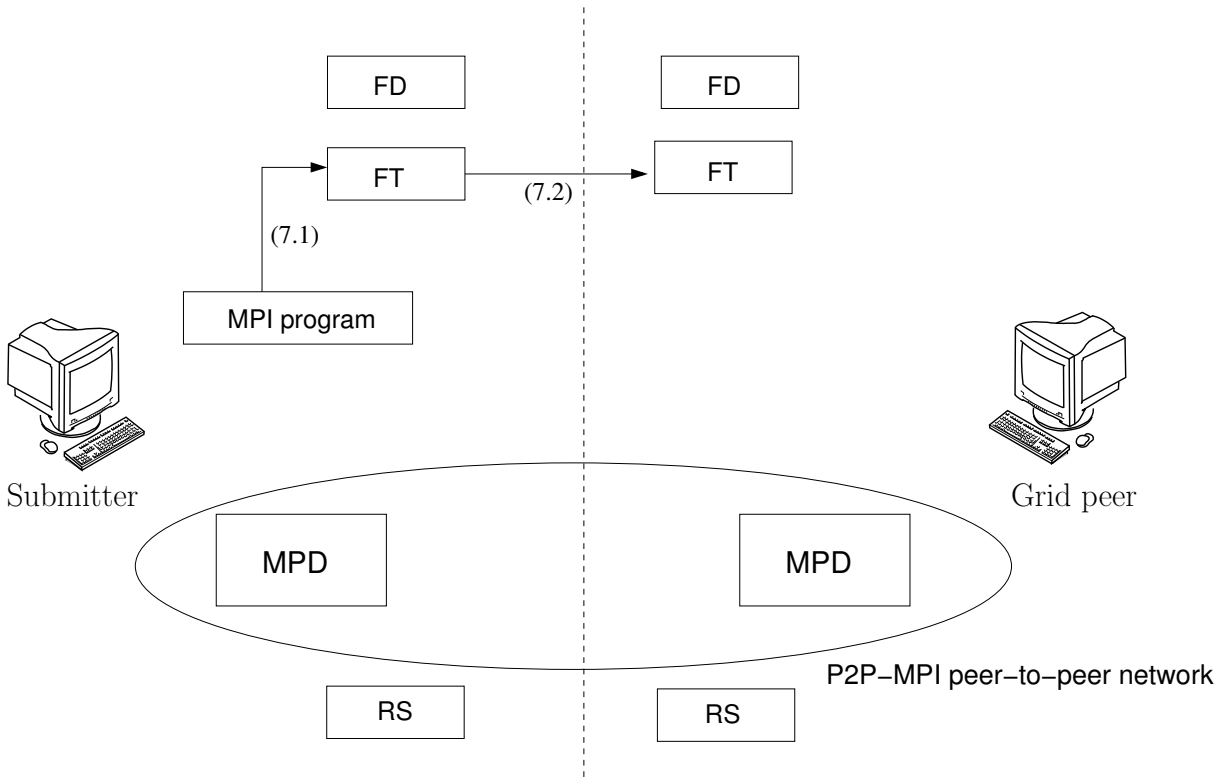
Registering: Local `MPD` contacts distant MPDs, give them MPI ranks, and IP, port of rank 0.

Scenario démarrage application



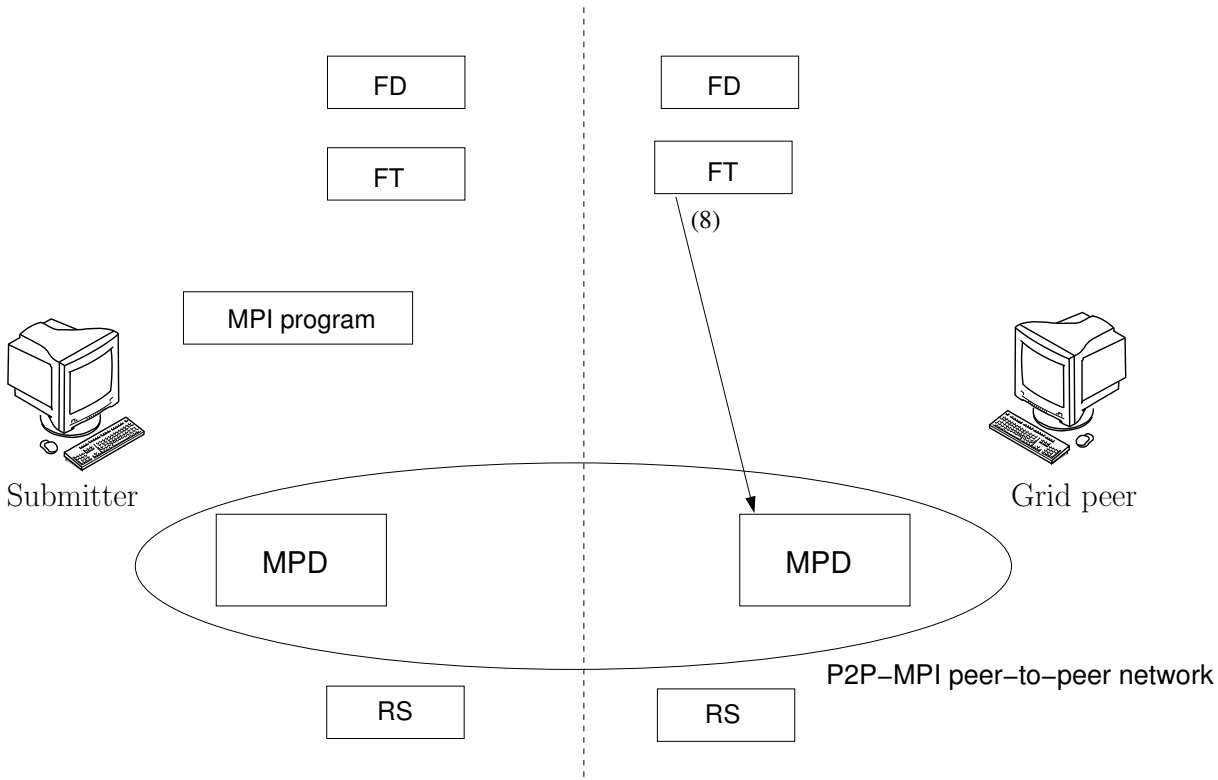
Hand-shake: The remote peers sends its `FD`, `FT` ports to rank 0.

Scenario démarrage application



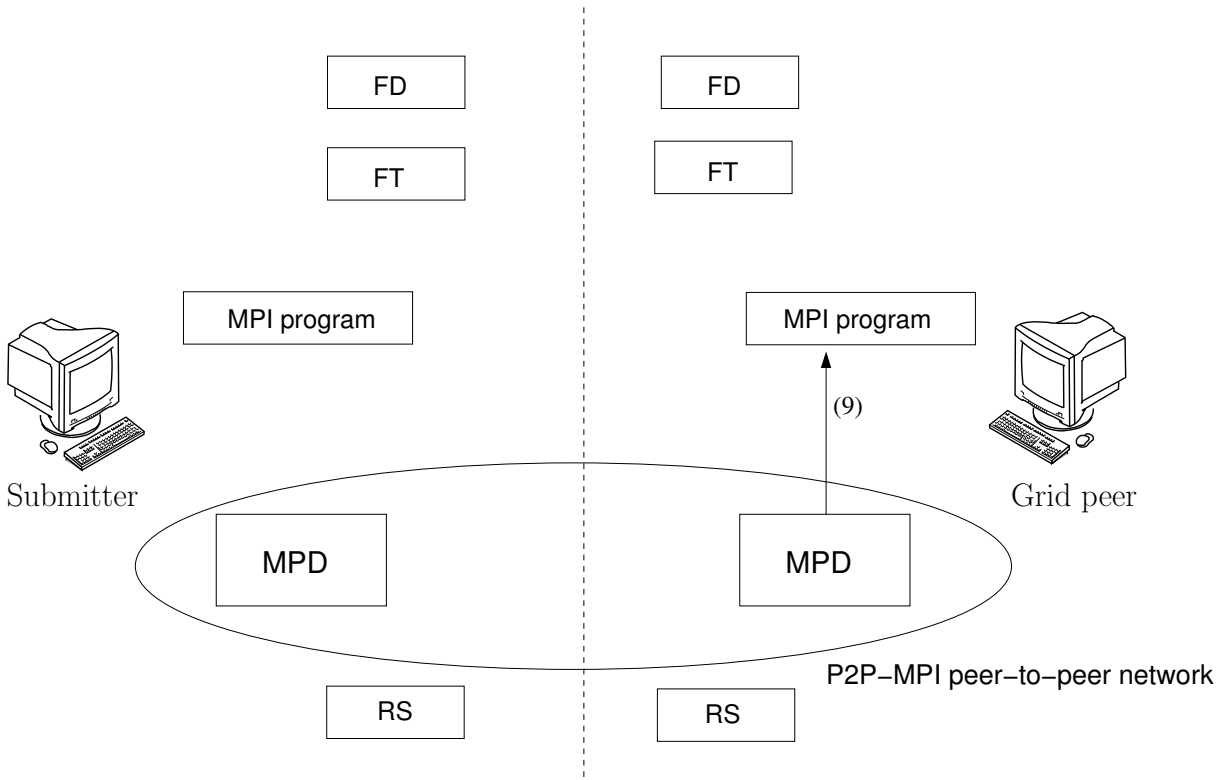
File staging: program and data transfer via `FT`.

Scenario démarrage application



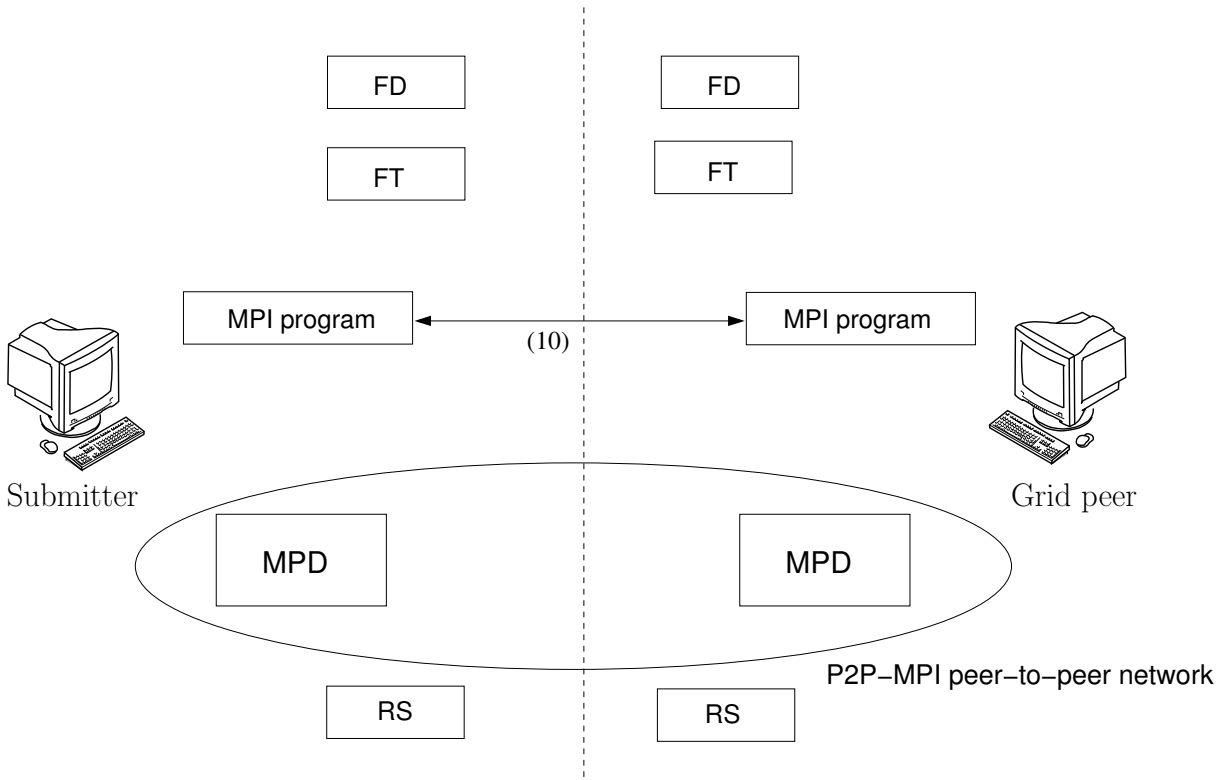
Execution Notification: `FD` notifies `MPD` to execute the transferred program.

Scenario démarrage application



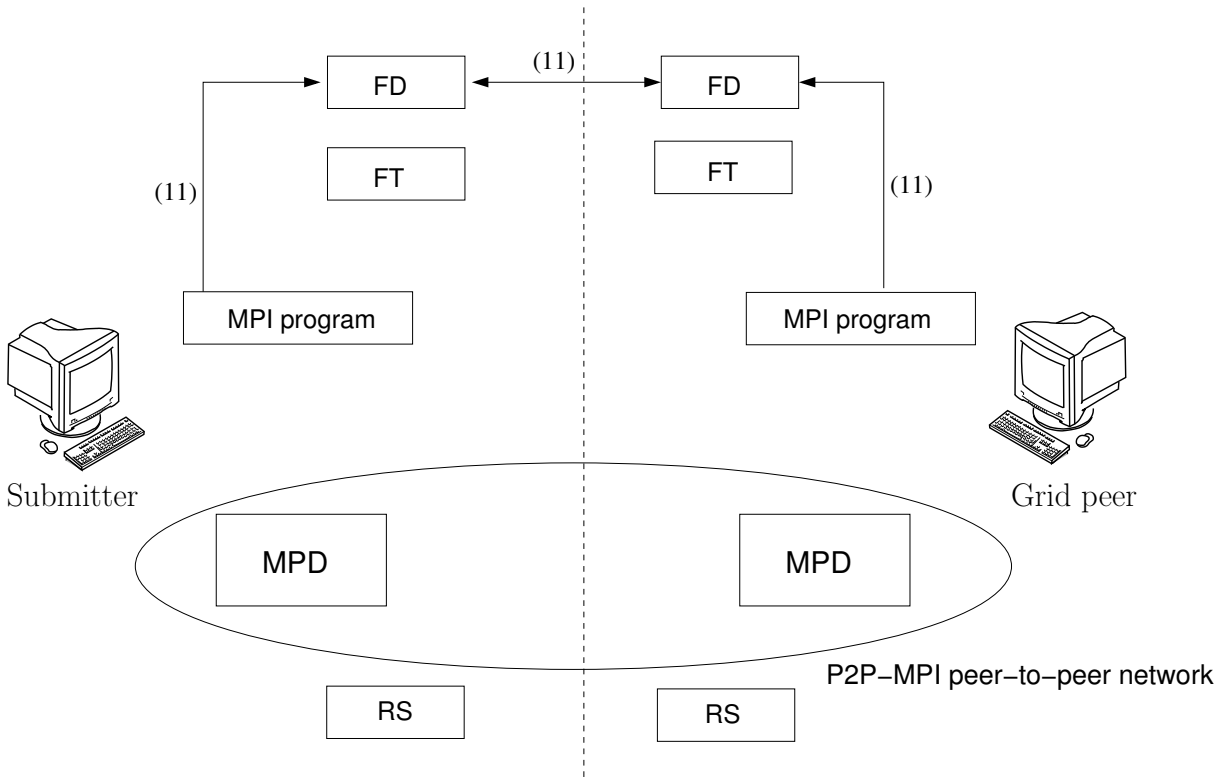
Remote executable lauch: `MPD` executes the transferred program.

Scenario démarrage application



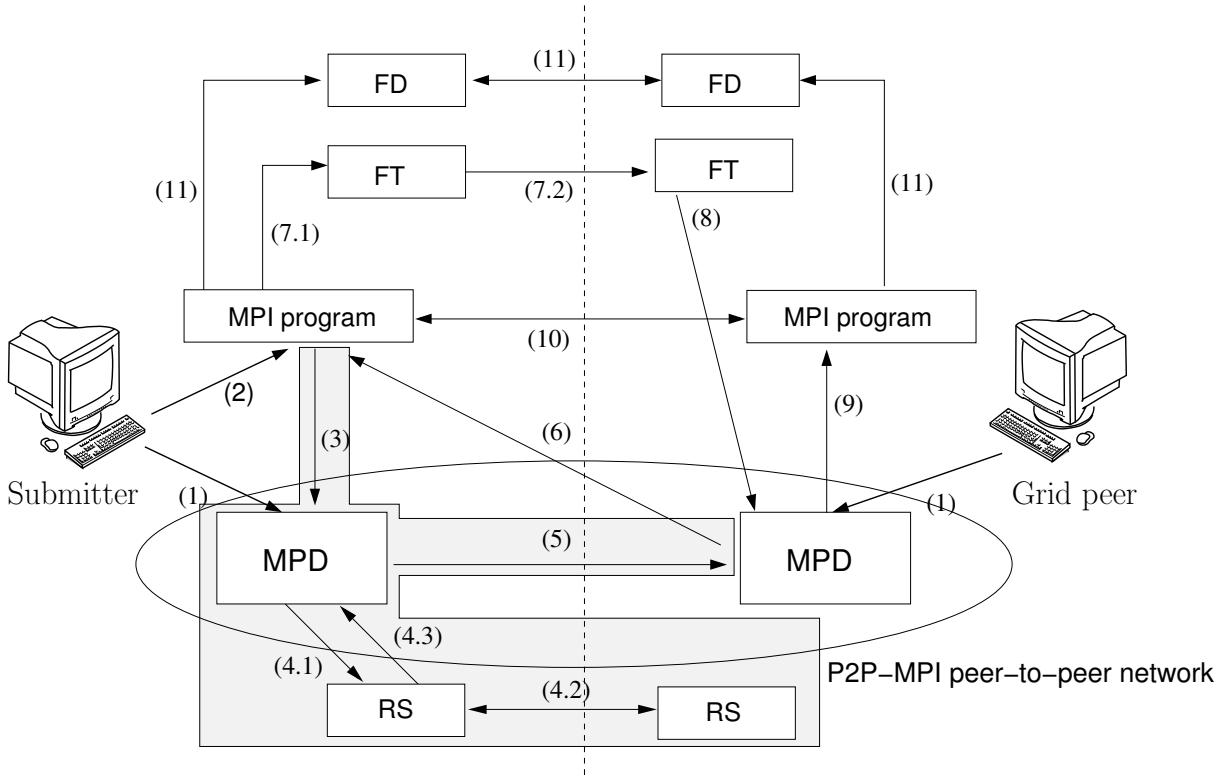
Execution preamble: spawn processes return their IP, port, rank to build the MPI communicator.

Scenario démarrage application



Fault detection: MPI processes register itself to `FD` for monitoring failure during the execution.

Scenario démarrage application



Problématique

Notre choix: tolérance aux fautes transparente aux applications et (presque) à l'utilisateur, pour un déploiement simple.

Le système repose sur deux "piliers" :

- Tolérance aux fautes : par redondance (réplication) des traitements.
- Détection des fautes : monitoring externe et distribué chargé d'informer des pannes.

Objectif : intégrer ces mécanismes dans un système extensible, prédictible, et fiable.

Problématique

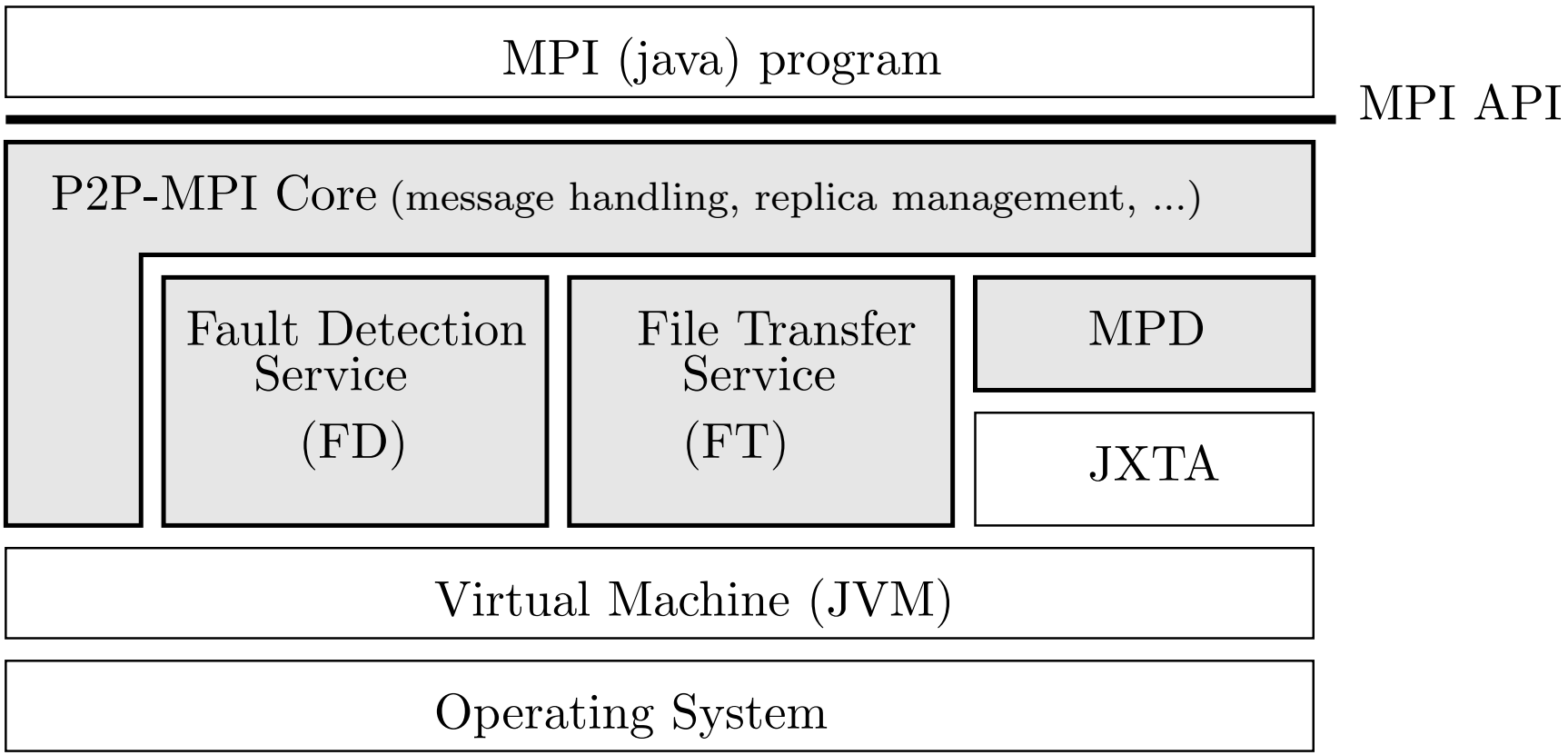
Notre choix: tolérance aux fautes transparente aux applications et (presque) à l'utilisateur, pour un déploiement simple.

Le système repose sur deux "piliers" :

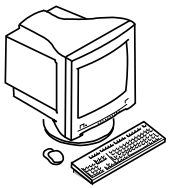
- Tolérance aux fautes : par redondance (réplication) des traitements.
- Détection des fautes : monitoring externe et distribué chargé d'informer des pannes.

Objectif : intégrer ces mécanismes dans un système extensible, prédictible, et fiable.

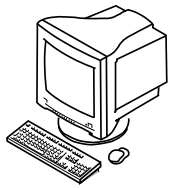
Un noeud P2P-MPI



Constitution d'une plate-forme pour une exécution

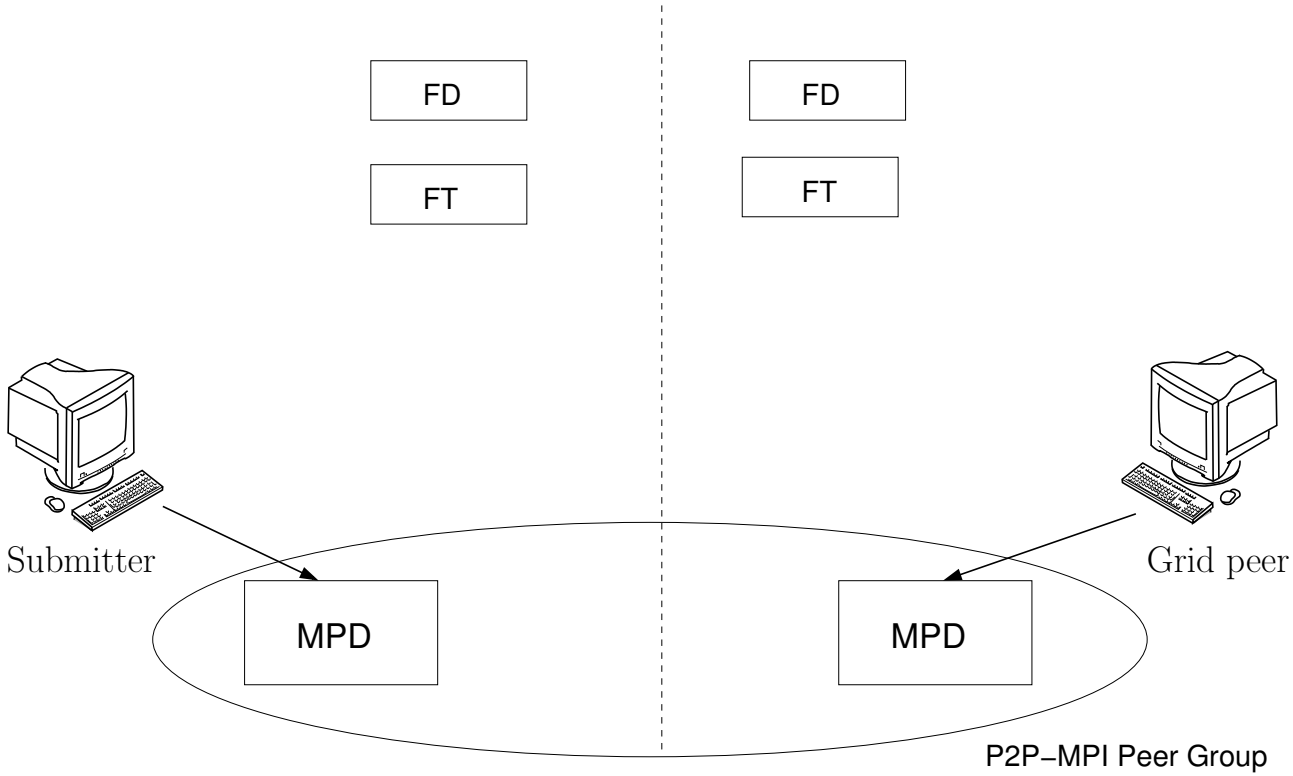


Submitter



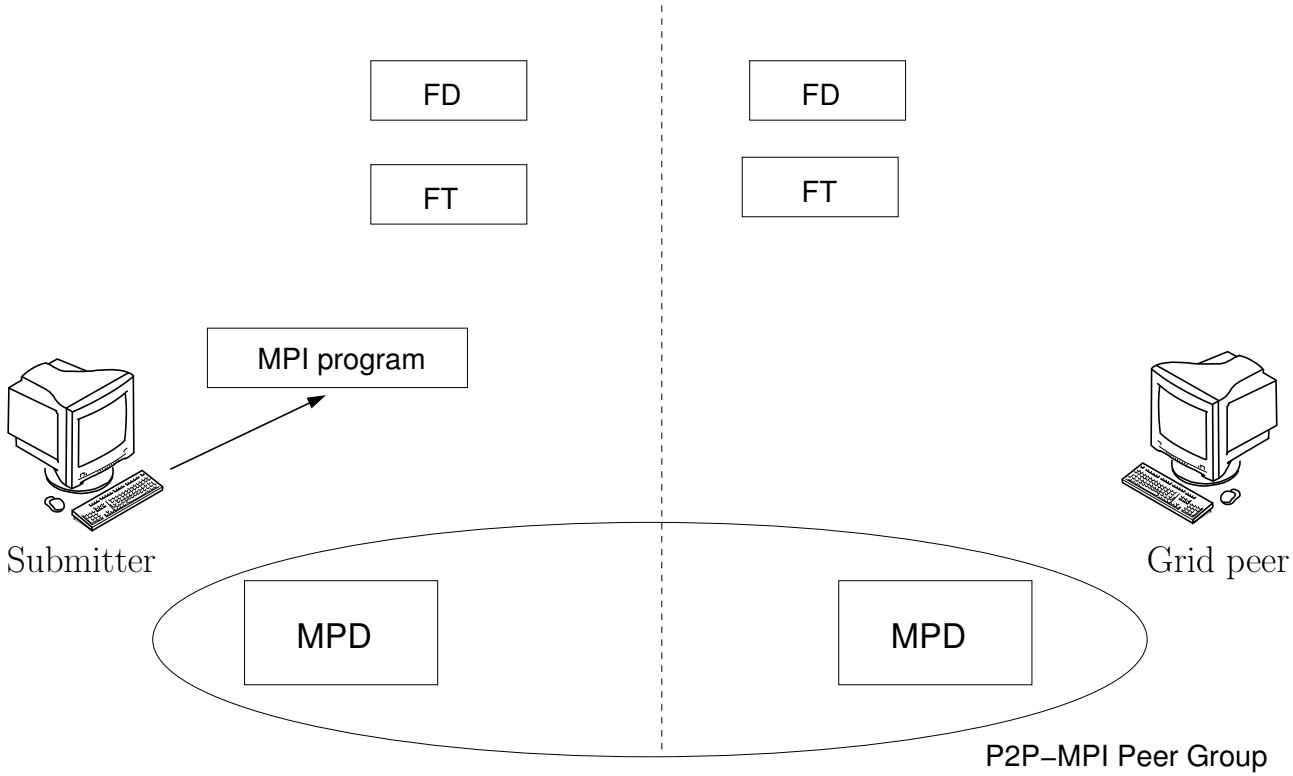
Grid peer

Constitution d'une plate-forme pour une exécution



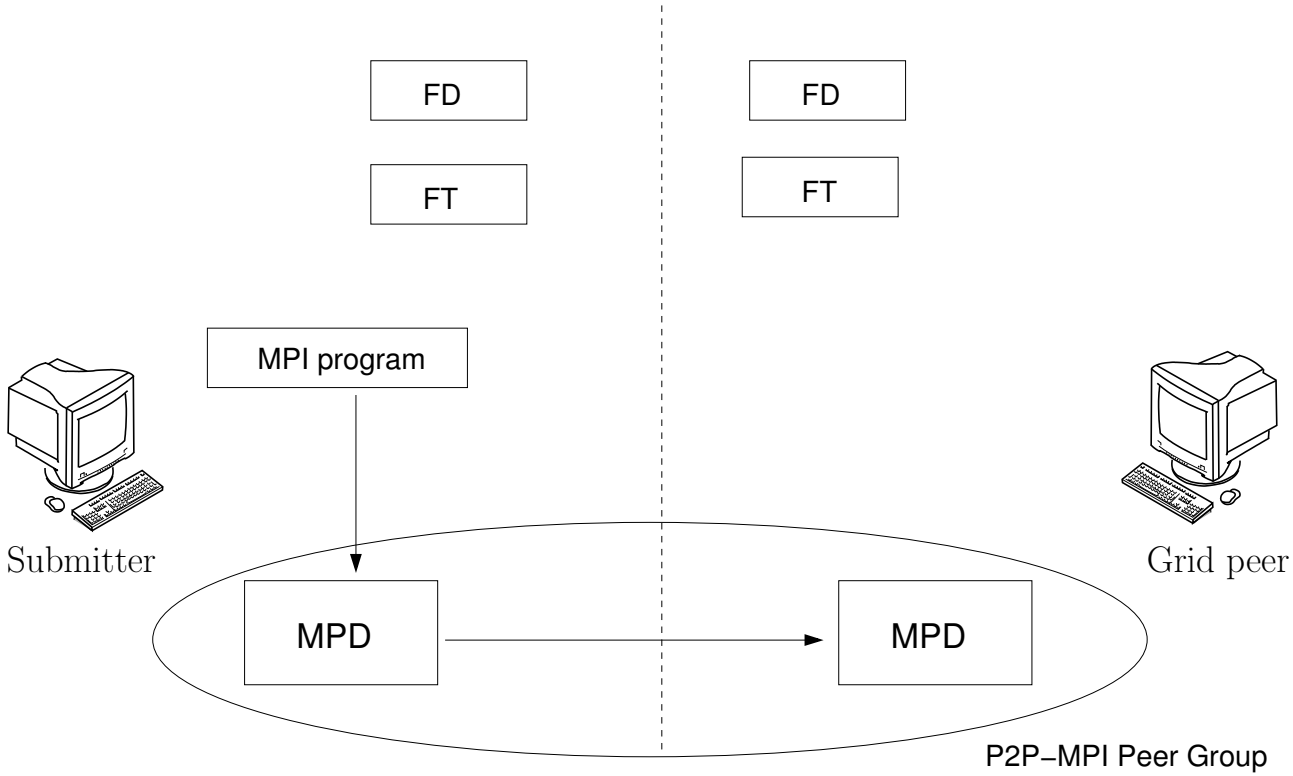
Inscription Groupe : `mpiboot` : MPD joint le groupe et publie son annonce

Constitution d'une plate-forme pour une exécution



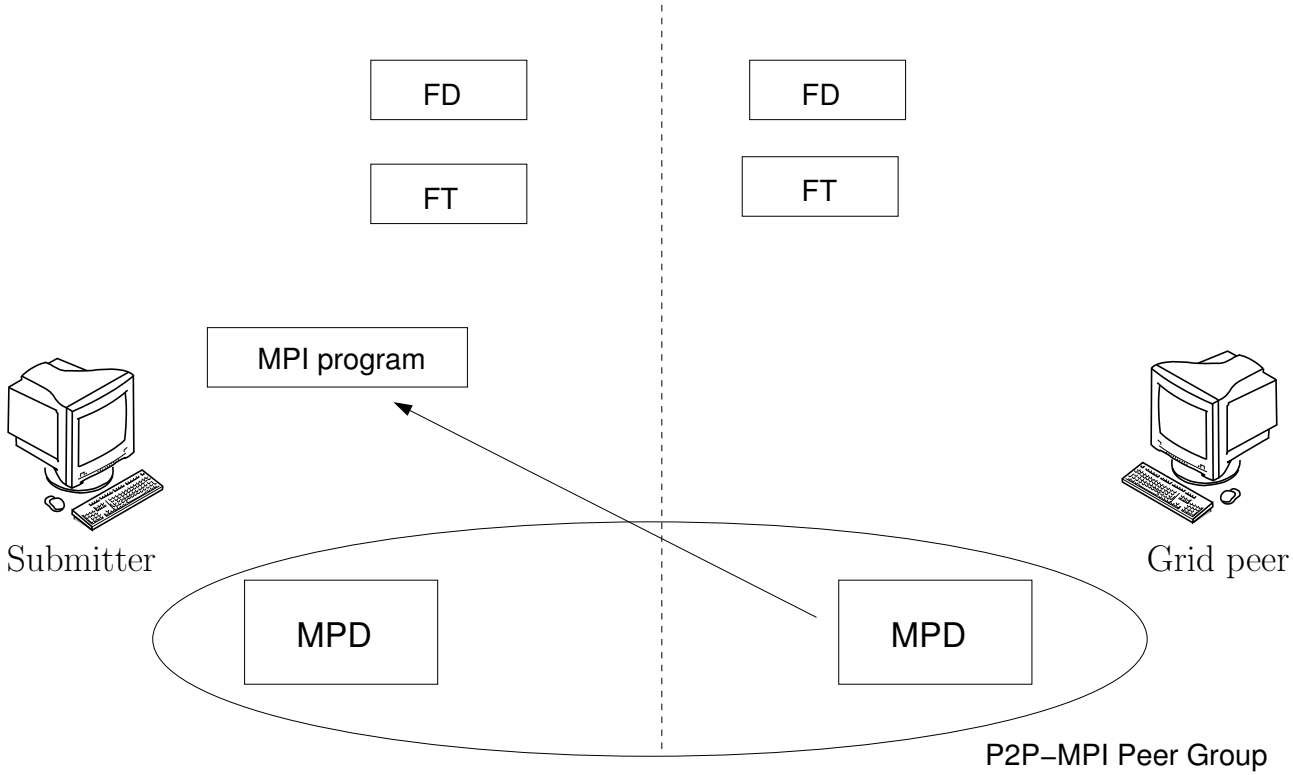
Requête utilisateur: `p2pmpirun -n 5 -r 2 -l filelist program`

Constitution d'une plate-forme pour une exécution



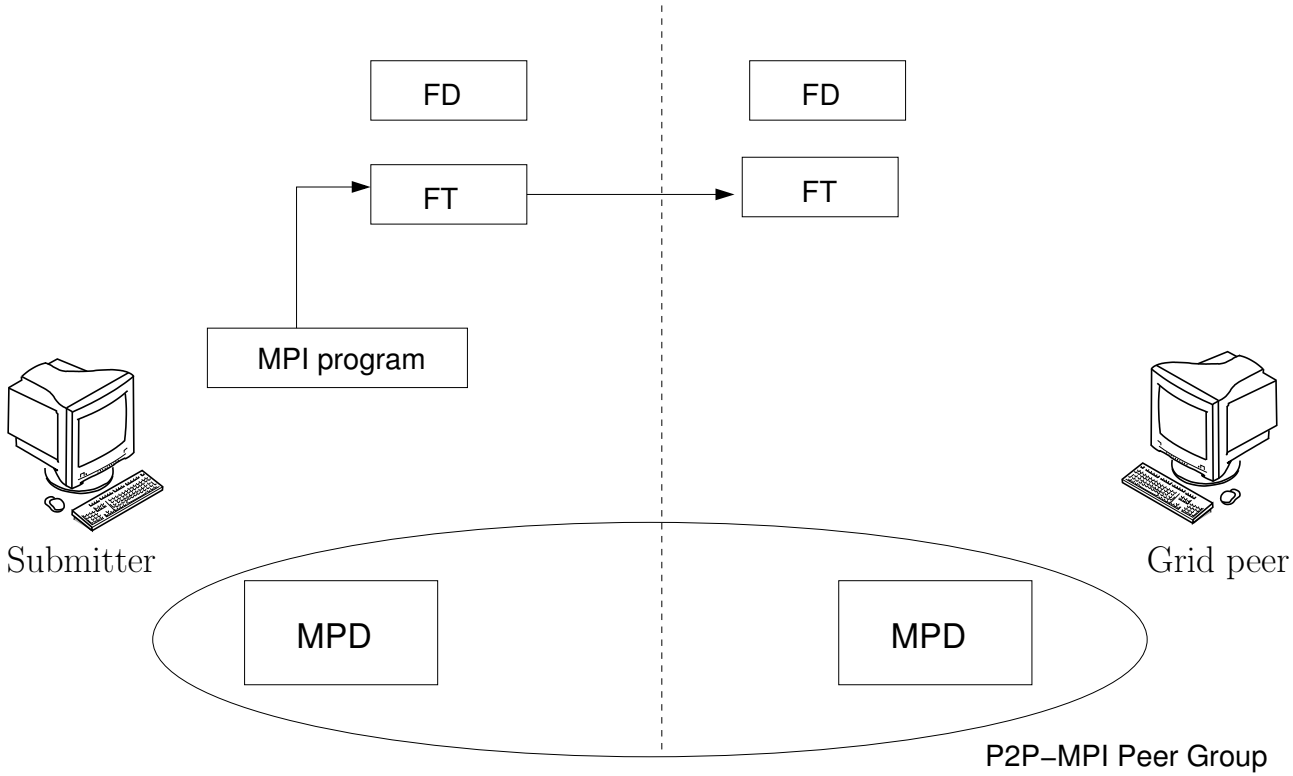
Recherche : accueil des ressources candidates, numérotation (rank) et diffusion du port application

Constitution d'une plate-forme pour une exécution



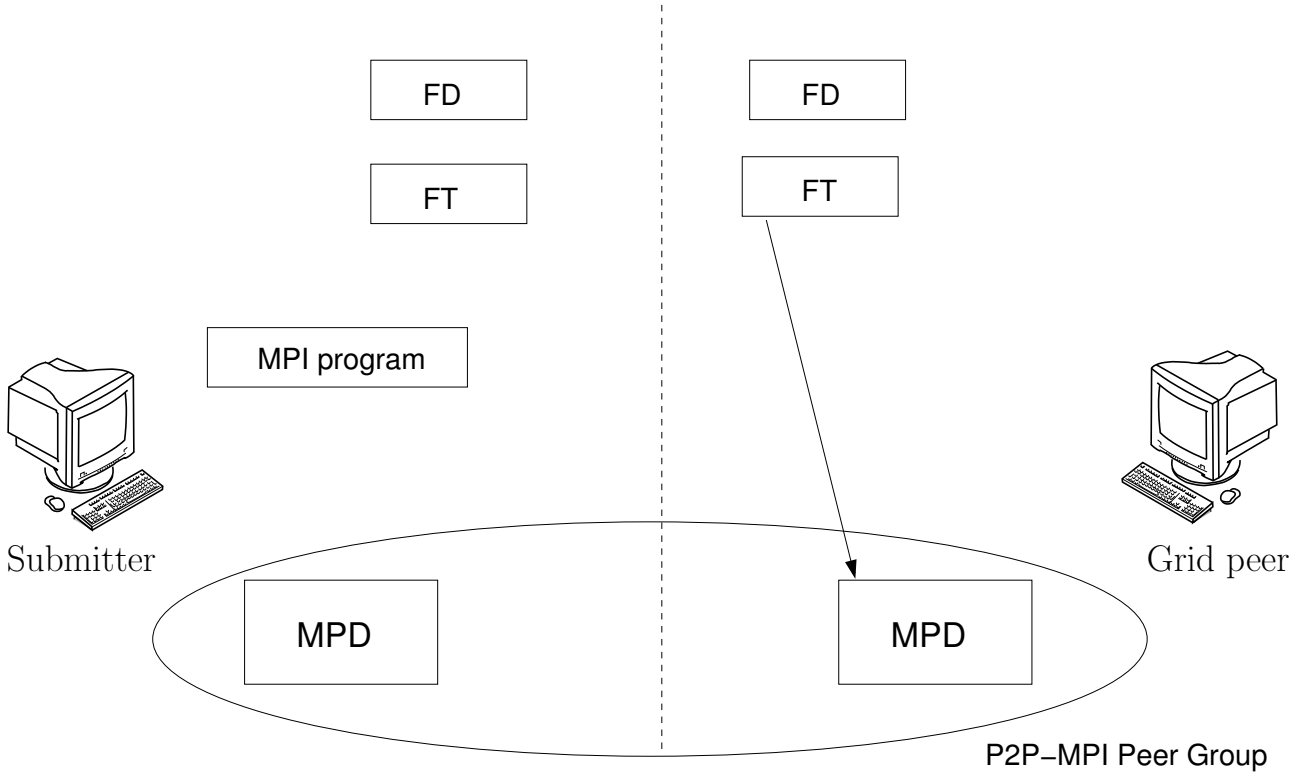
Reconnaissance : candidat retourne les ports de ses services FT et FD

Constitution d'une plate-forme pour une exécution



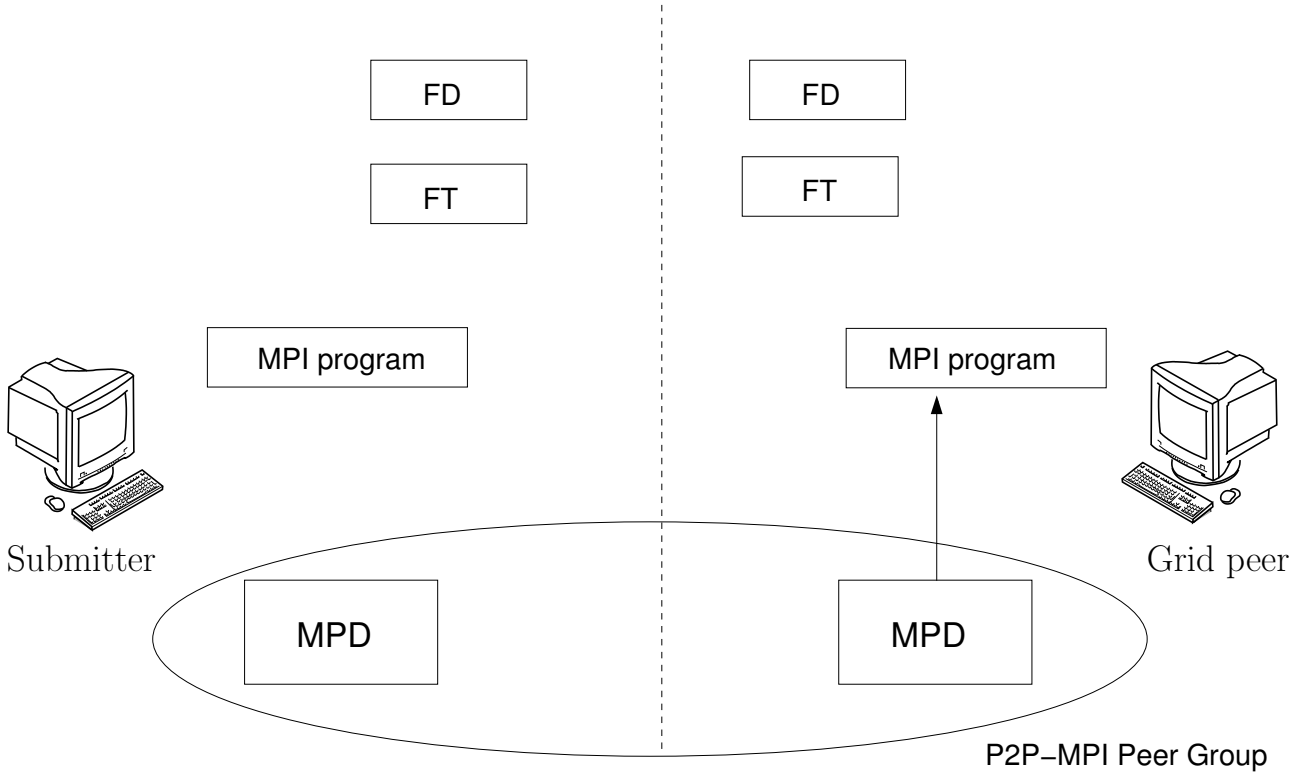
Téléchargement : programmes et données transférées entre noeuds via services FT

Constitution d'une plate-forme pour une exécution



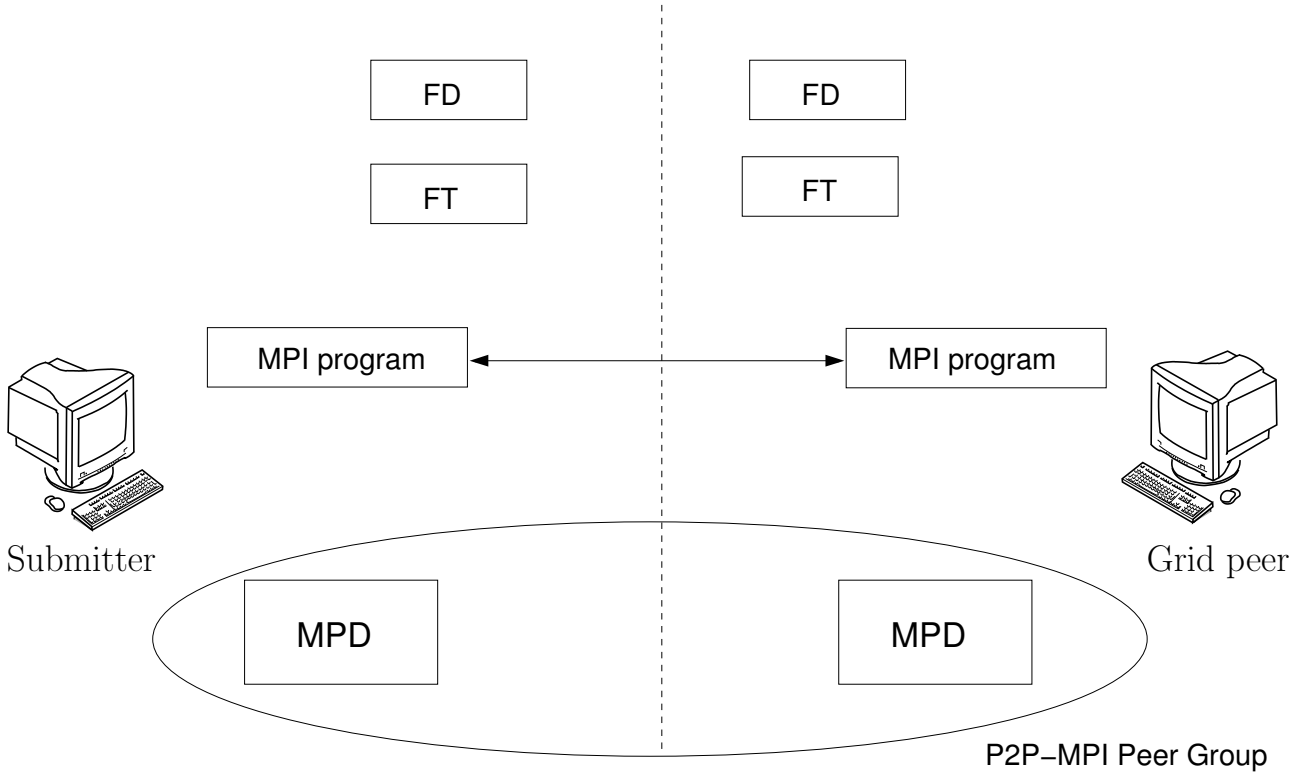
Notification : FT indique au MPD d'exécuter le programme

Constitution d'une plate-forme pour une exécution



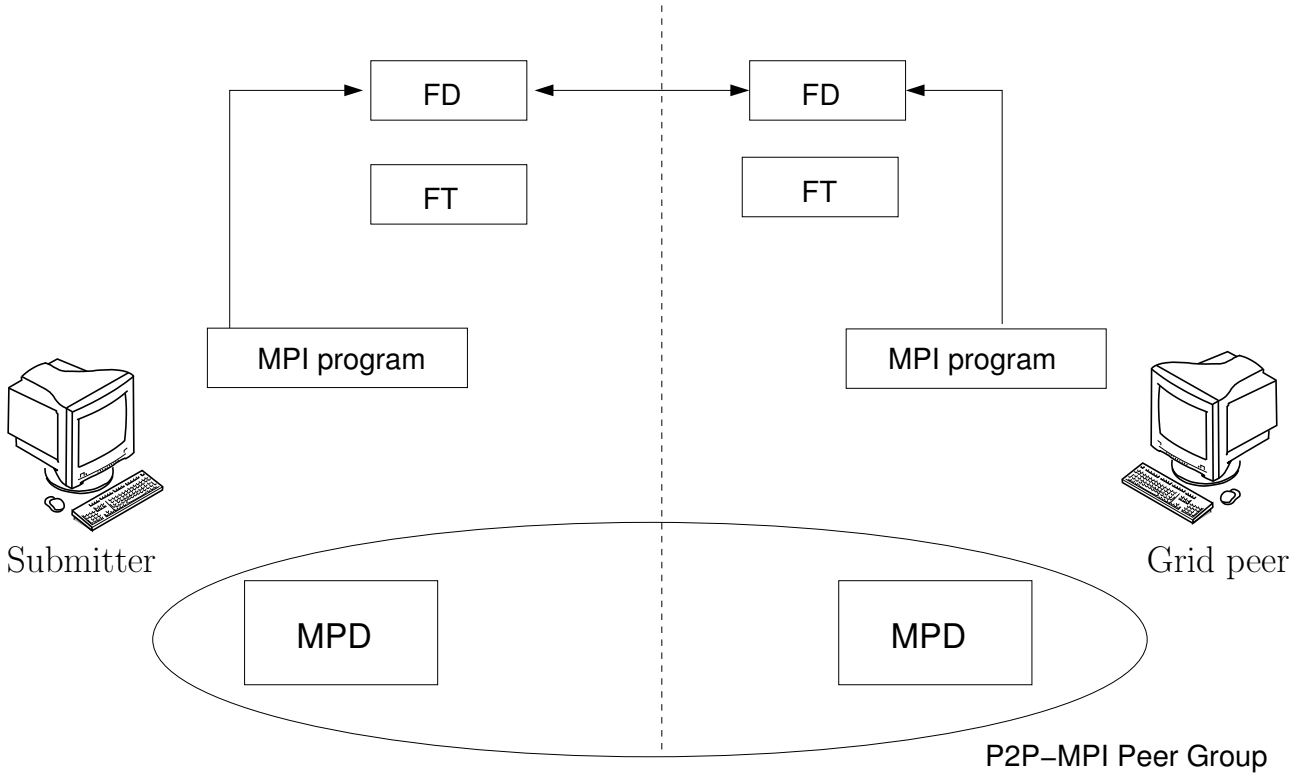
Exécution : MPD démarre l'application MPI

Constitution d'une plate-forme pour une exécution



Construction plate-forme d'exécution : MPIs s'échangent leur IP et leur Port

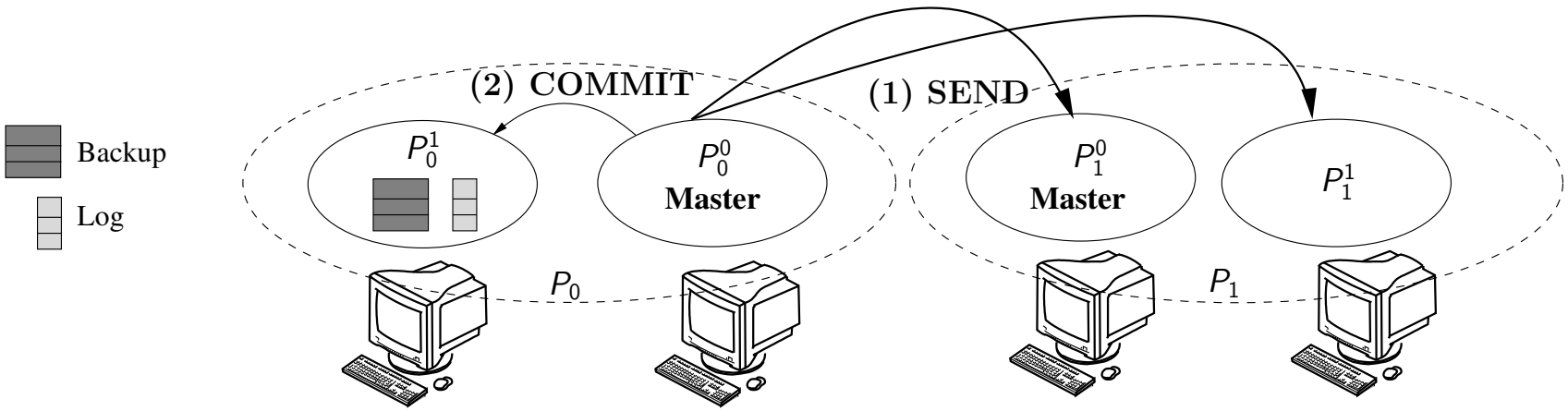
Constitution d'une plate-forme pour une exécution



Gestion pannes : les applications s'enregistrent dans le service de détection de panne

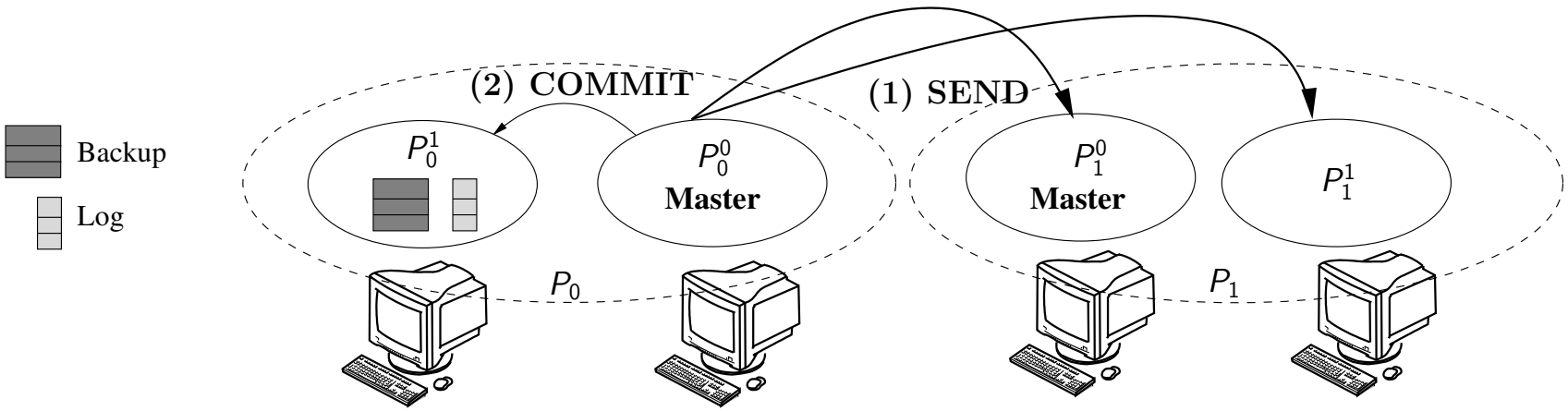
Réplication

- Combien de réplica pour chaque processus est indiqué par l'utilisateur (option -r).
- Garantie : pas 2 copies d'un processus sur la même machine.
- Réplication est transparente aux programmeurs (Send $P_0 \rightarrow P_1$).



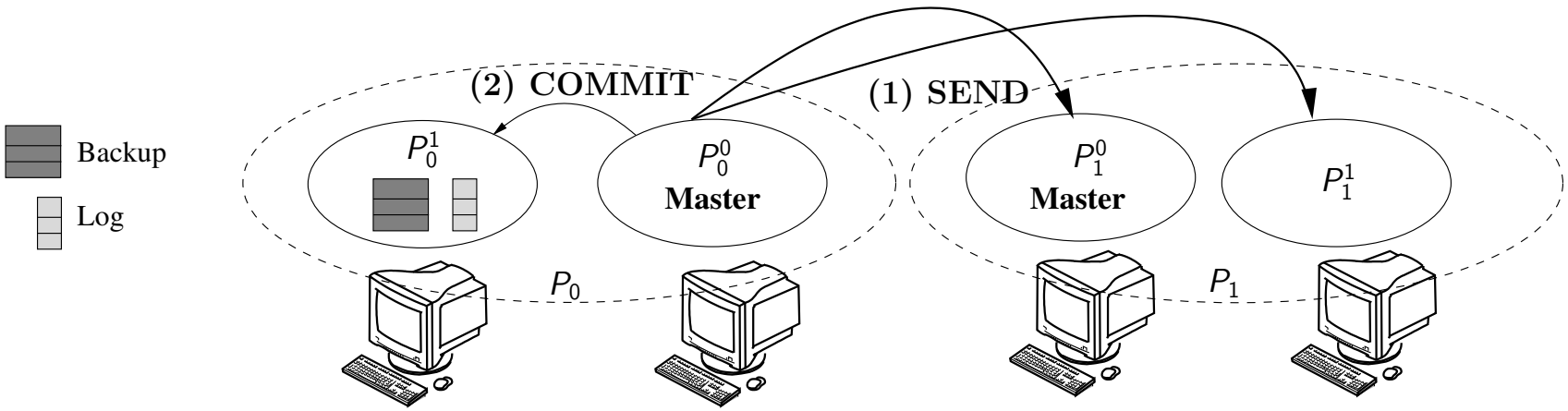
Réplication

- Combien de réplica pour chaque processus est indiqué par l'utilisateur (option -r).
- Garantie : pas 2 copies d'un processus sur la même machine.
- Réplication est transparente aux programmeurs (Send $P_0 \rightarrow P_1$).

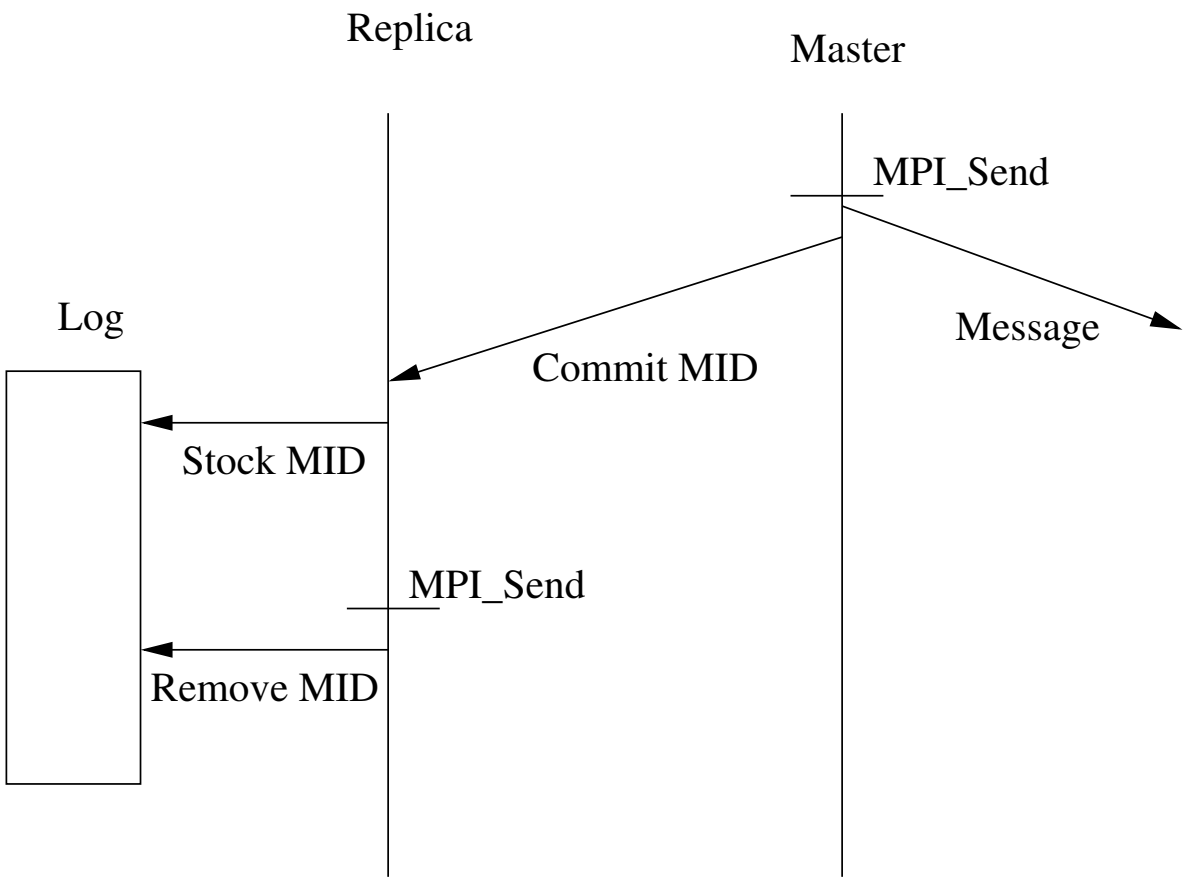


Réplication

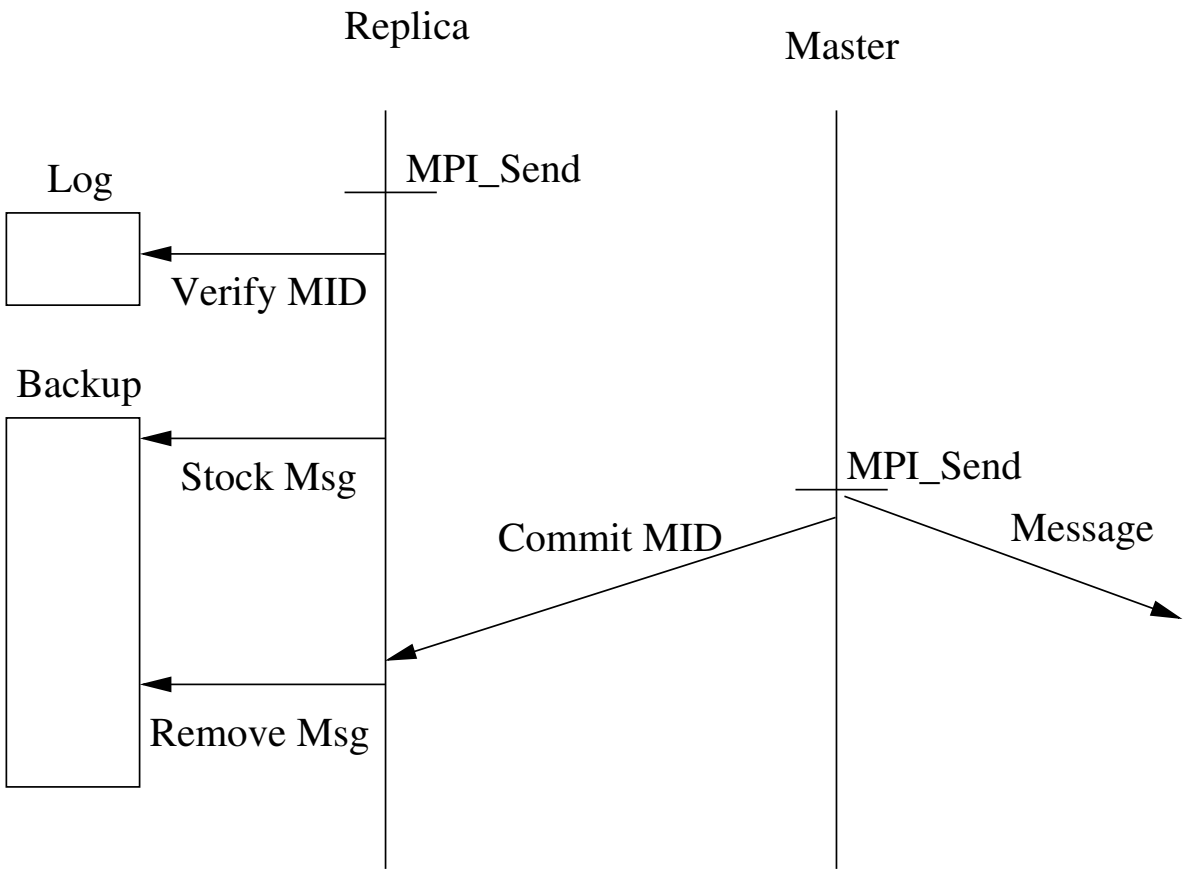
- Combien de réplica pour chaque processus est indiqué par l'utilisateur (option -r).
- Garantie : pas 2 copies d'un processus sur la même machine.
- Réplication est transparente aux programmeurs (Send $P_0 \rightarrow P_1$).



Scénario master arrive au MPI_SEND avant réplica



Scénario réplica arrive au MPI_SEND avant master



Cas de panne

Lorsqu'il y a une panne

- Panne du maître :
  - Replica choisit un nouveau maître.
  - Le nouveau maître envoie tous les messages qui reste dans sa table de backup
- Panne du/des replica :
  - Il n'y a rien à faire. L'exécution continue normalement.

Cas de panne

Lorsqu'il y a une panne

- Panne du maître :
  - Replica choisit un nouveau maître.
  - Le nouveau maître envoie tous les messages qui reste dans sa table de backup
- Panne du/des replica :
  - Il n'y a rien à faire. L'exécution continue normalement.

Cas de panne

Lorsqu'il y a une panne

- Panne du maître :
  - Replica choisit un nouveau maître.
  - Le nouveau maître envoie tous les messages qui reste dans sa table de backup
- Panne du/des replica :
  - Il n'y a rien à faire. L'exécution continue normalement.

Probabilité de panne d'une application

Hyp: dans une unité de temps, chaque processus à une probabilité f de tomber en panne.

Probabilité qu'une application à n processus, sans réplication, tombe en panne ?

$\Leftrightarrow$ Probabilité que 1, ou 2, ou ... n processus tombent en panne
$\Leftrightarrow$ 1 - (probabilité qu'aucun processus tombe en panne)
$\Leftrightarrow$ $1 - (1 - f)^n$

Probabilité de panne d'une application

Hyp: dans une unité de temps, chaque processus à une probabilité f de tomber en panne.

Probabilité qu'une application à n processus, sans réplication, tombe en panne ?

$\Leftrightarrow$ Probabilité que 1, ou 2, ou ... n processus tombent en panne
$\Leftrightarrow$ 1 - (probabilité qu'aucun processus tombe en panne)
$\Leftrightarrow$ $1 - (1 - f)^n$

Probabilité de panne d'une application

Hyp: dans une unité de temps, chaque processus à une probabilité f de tomber en panne.

Probabilité qu'une application à n processus, taux réplication r, tombe en panne ?

Un processus répliqué tombe en panne ssi ses r copies tombent en panne. Probabilité : f^r

Probabilité qu'une application à n processus tombe en panne
$\Leftrightarrow$ Probabilité que 1, ou 2, ou ... n processus répliqués tombent en pa
$\Leftrightarrow$ 1 - (probabilité qu'aucun processus répliqué ne tombe en panne)
$\Leftrightarrow$ $1-(1-f^r)^n$

Probabilité de panne d'une application

Hyp: dans une unité de temps, chaque processus à une probabilité f de tomber en panne.

Probabilité qu'une application à n processus, taux réplication r, tombe en panne ?

Un processus répliqué tombe en panne ssi ses r copies tombent en panne. Probabilité : f^r

Probabilité qu'une application à n processus tombe en panne
$\Leftrightarrow$ Probabilité que 1, ou 2, ou ... n processus répliqués tombent en pa
$\Leftrightarrow$ 1 - (probabilité qu'aucun processus répliqué ne tombe en panne)
$\Leftrightarrow$ $1 - (1 - f^r)^n$

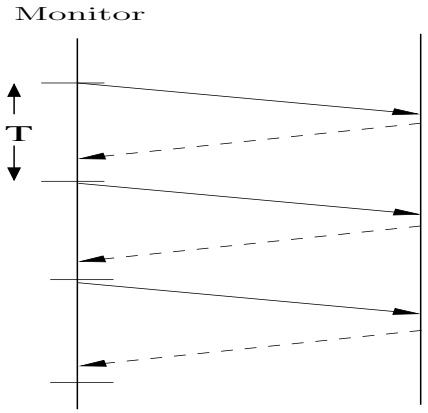
Probabilité de panne d'une application

Exemples numériques

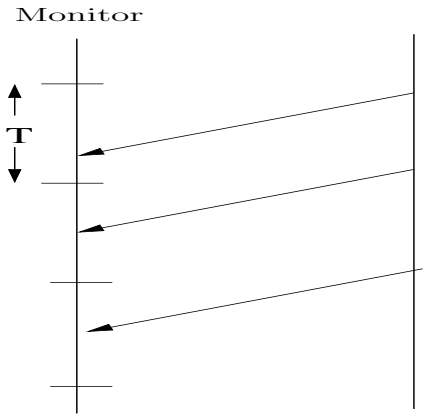
n	f	r	*Prob.*
8	5%	1	0.33
8	5%	2	0.01
50	5%	1	0.92
50	5%	2	0.11
100	5%	1	0.99
100	5%	2	0.22

Mais comment peut-on détecter une panne ?

Protocole de détection des pannes (Centralisé)



Pull Model

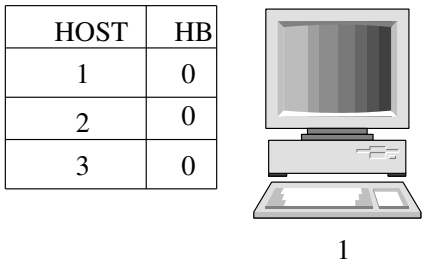


Push Model

L'inconvénient :

- Besoin d'un serveur (monitor) centralisé
- Goulot d'étranglement (réseau) sur serveur

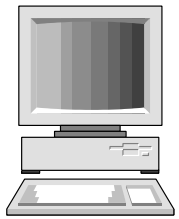
Protocole Gossip (Distribué)



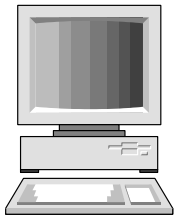
Chaque nœud maintient une table qui contient le dernier heartbeat des autres.

Protocole Gossip (Distribué)

HOST	HB
1	1
2	0
3	0

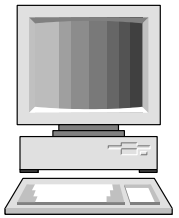


1



2

HOST	HB
1	0
2	0
3	0

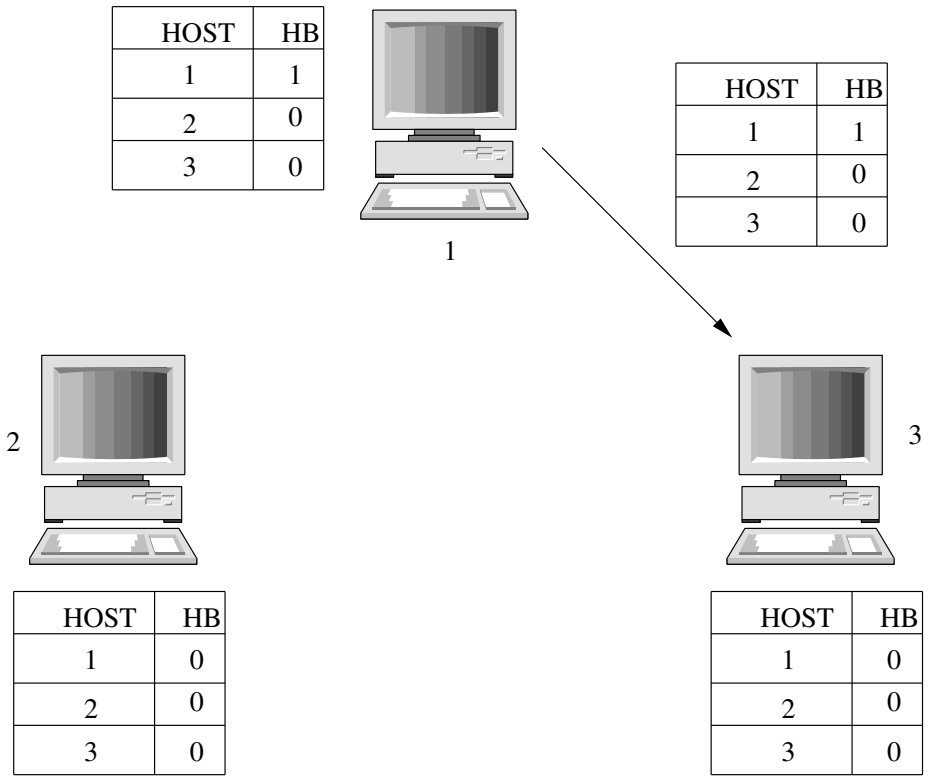


3

HOST	HB
1	0
2	0
3	0

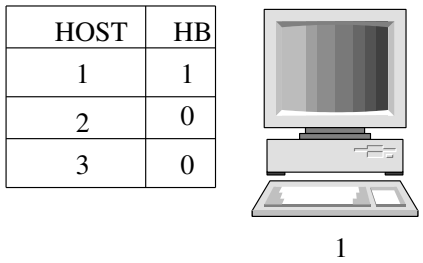
Périodiquement, Un nœud augmente son heartbeat

Protocole Gossip (Distribué)



Après avoir augmenté son heartbeat, il envoie sa table à un autre noeud.

Protocole Gossip (Distribué)



Il fusionne sa table avec une table qu'il a reçu (garde l'heartbeat maximum)

Protocole Gossip (Distribué)

La détection : Chaque nœud peut détecter une panne individuellement en vérifiant si l'heartbeat d'un nœud n'a pas augmenté depuis un certain temps.

Avantages :

- Distribution de la charge sur le réseau
- Pas de serveur centralisé

Mot clé :

- T_{gossip} est la période d'envoi de la table d'heartbeat
- $T_{cleanup}$ est le temps de détection et de suppression d'un nœud défaillant de la table

$T_{cleanup} = N_{round} * T_{gossip}$

Protocole Gossip (Distribué)

La détection : Chaque nœud peut détecter une panne individuellement en vérifiant si l'heartbeat d'un nœud n'a pas augmenté depuis un certain temps.

Avantages :

- Distribution de la charge sur le réseau
- Pas de serveur centralisé

Mot clé :

- T_{gossip} est la période d'envoi de la table d'heartbeat
- $T_{cleanup}$ est le temps de détection et de suppression d'un noeud défaillant de la table
- $T_{cleanup} = N_{round} \times T_{gossip}$

Protocole Gossip (Distribué)

La détection : Chaque nœud peut détecter une panne individuellement en vérifiant si l'heartbeat d'un nœud n'a pas augmenté depuis un certain temps.

Avantages :

- Distribution de la charge sur le réseau
- Pas de serveur centralisé

Mot clé :

- T_{gossip} est la période d'envoi de la table d'heartbeat
- $T_{cleanup}$ est le temps de détection et de suppression d'un noeud défaillant de la table
- $T_{cleanup} = N_{round} \times T_{gossip}$

Protocole Gossip (Distribué)

La détection : Chaque nœud peut détecter une panne individuellement en vérifiant si l'heartbeat d'un nœud n'a pas augmenté depuis un certain temps.

Avantages :

- Distribution de la charge sur le réseau
- Pas de serveur centralisé

Mot clé :

- T_{gossip} est la période d'envoi de la table d'heartbeat
- $T_{cleanup}$ est le temps de détection et de suppression d'un noeud défaillant de la table
- $T_{cleanup} = N_{round} \times T_{gossip}$

Protocole Gossip (Distribué)

La détection : Chaque nœud peut détecter une panne individuellement en vérifiant si l'heartbeat d'un nœud n'a pas augmenté depuis un certain temps.

Avantages :

- Distribution de la charge sur le réseau
- Pas de serveur centralisé

Mot clé :

- T_{gossip} est la période d'envoi de la table d'heartbeat
- $T_{cleanup}$ est le temps de détection et de suppression d'un noeud défaillant de la table
- $T_{cleanup} = N_{round} \times T_{gossip}$

Aléatoire (Random Gossip)

Fonction :

- Le design d'origine du protocole gossip [Van Renesse 97]
- Choisit un noeud au hasard, et envoie un message gossip

Problème :

- Temps de détection non-déterministe
- N_{round} petit, provoque fausse détection (si un noeud n'a pas reçu un message gossip)
- N_{round} grand, la détection est lente

Aléatoire (Random Gossip)

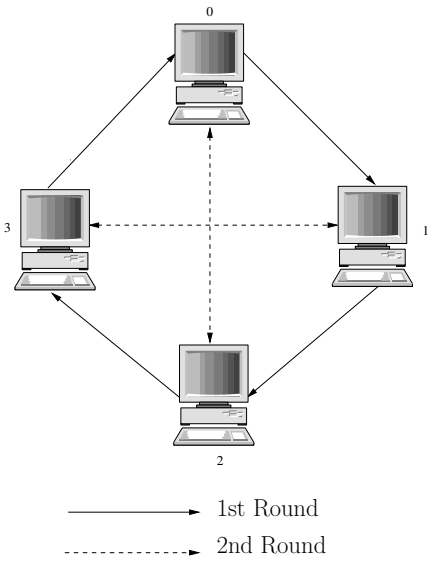
Fonction :

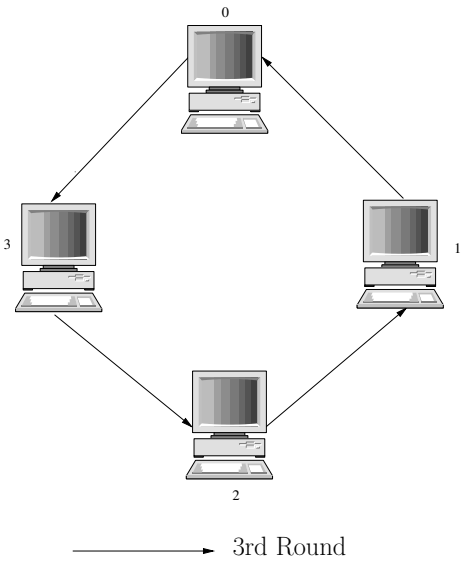
- Le design d'origine du protocole gossip [Van Renesse 97]
- Choisit un noeud au hasard, et envoie un message gossip

Problème :

- Temps de détection non-déterministe
- N_{round} petit, provoque fausse détection (si un noeud n'a pas reçu un message gossip)
- N_{round} grand, la détection est lente

Round-Robin

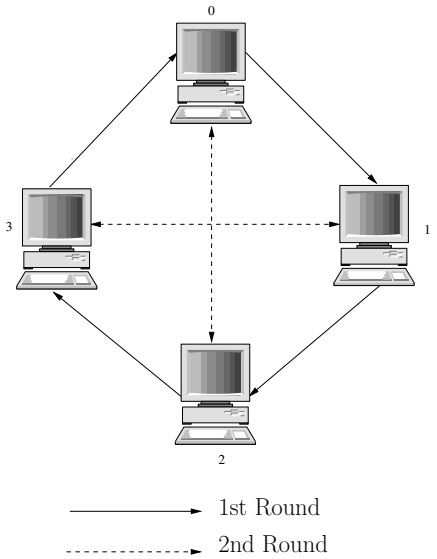




Propriété :

- $d = (s + r) \mod n, \quad 1 \leq r < n$
- $N_{round} = n - 1$

Round-Robin Binaire



Propriété :

- $d = (s + 2^{r-1}) \mod n, \quad 1 \leq r \leq log_2(n)$
- $N_{round} = \lceil log_2(n) \rceil$

BRR VS Réplica du P2P-MPI

Exemple : n = 2, r = 2

Application :

- $P_a = 1 - (1 - f^2)^2 = 2f^2 - f^4$

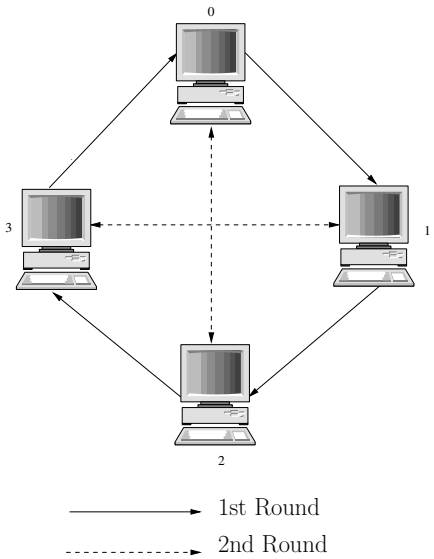
Binaire Round-Robin :

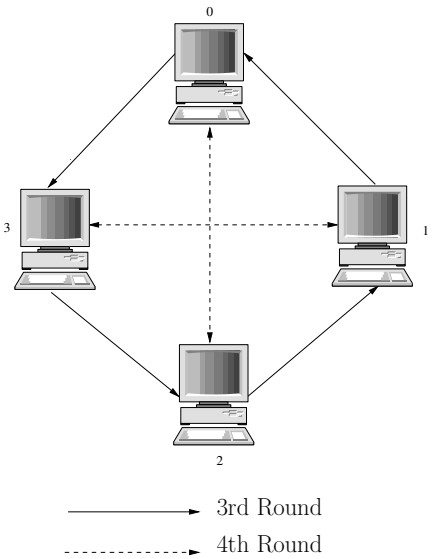
- panne 1 processus : $p = 0$
- panne 2 processus : $p = 4 * (f^2(1 - f)^2)$
- panne 3 processus : $p = 4 * (f^3(1 - f))$
- panne 4 processus : $p = f^4$
- Total: $P_{fd} = 4f^2 - 4f^3 + f^4$

Conclusion :

$f \in \ ]0, 1[\ \Rightarrow P_{fd} > P_a \ (f = 0.05 \Rightarrow P_a = 0.005, P_{fd} = 0.0095)$

Double Round-Robin Binaire





Propriété :

- $d = \begin{cases} (s + 2^{r-1}) \mod n & \text{if } 1 \leq r \leq log_2(n) \\ (s - 2^{r-log_2(n)-1}) \mod n & \text{if } log_2(n) < r \leq 2 * log_2(n) \end{cases}$
- $N_{round} = 2 * \lceil log_2(n) \rceil$

Difficulté de développement

Problème :

- Les nœuds n'ont pas d'horloge globale
- Utilisation, l'horloge logique pour gossip.
- Comment peut on être sûr que l'horloge logique démarre au même moment?

Solution:

- Réglage automatique de l'heartbeat

Difficulté de développement

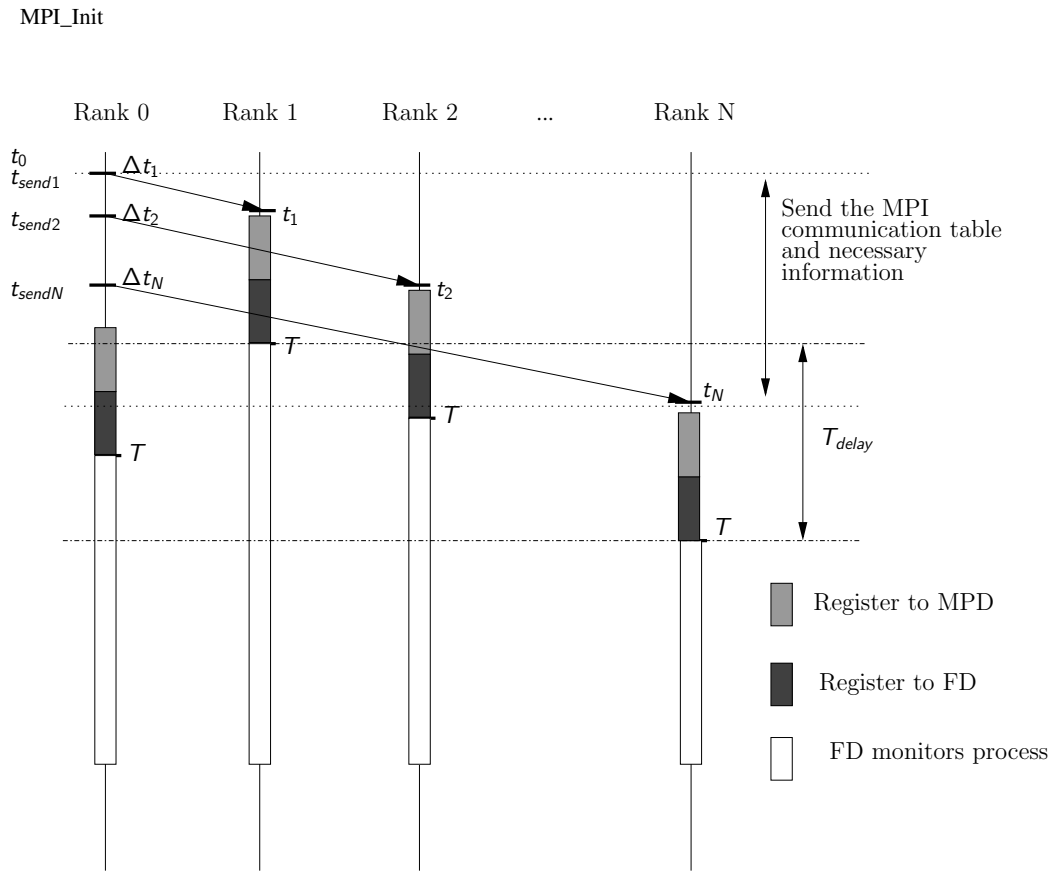
Problème :

- Les nœuds n'ont pas d'horloge globale
- Utilisation, l'horloge logique pour gossip.
- Comment peut on être sûr que l'horloge logique démarre au même moment?

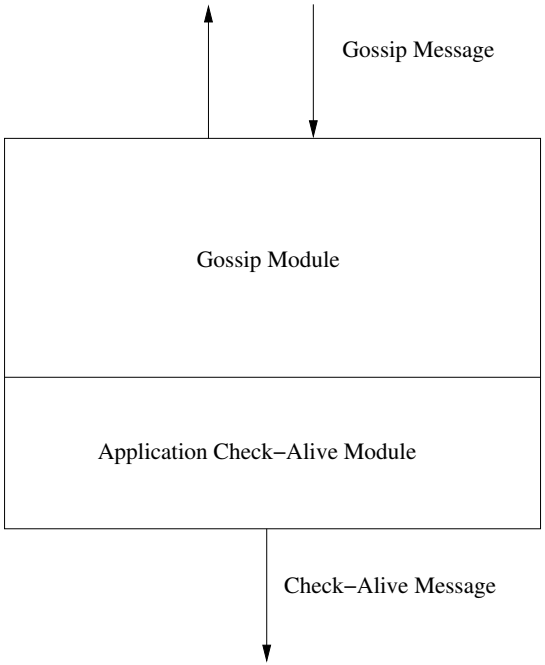
Solution:

- Réglage automatique de l'heartbeat

Réglage l'heartbeat



Module FD en détails



Pannes d'un noeud

Problème :

- Crash d'un noeud
- Coupure de réseau

Solution dans P2P-MPI :

Grace à FD, après $T_{cleanup}$ $(2 * log_2(N) \times T_{gossip})$, Le FD peut détecter qu'il y a un noeud qui n'augmente pas son heartbeat depuis. Donc, il notifie l'application MPI pour traiter (Choisi nouveau maître, si c'est le maître qui crash)

Pannes d'un noeud

Problème :

- Crash d'un noeud
- Coupure de réseau

Solution dans P2P-MPI :

Grace à FD, après $T_{cleanup}$ $(2 * log_2(N) \times T_{gossip})$, Le FD peut détecter qu'il y a un noeud qui n'augmente pas son heartbeat depuis. Donc, il notifie l'application MPI pour traiter (Choisi nouveau maître, si c'est le maître qui crash)

Pannes de l'application

Problème :

- application provoque une faute (e.g. divide by zero)

Solution dans P2P-MPI :

Le module *check-alive* dans FD va détecter une panne d'application si l'application ne répond pas au message *check-alive*. Après détection de la panne, il arrête de fonctionner (d'envoyer son heartbeat). Les autres noeuds peuvent alors détecter la panne comme la panne d'un noeud.

Pannes de l'application

Problème :

- application provoque une faute (e.g. divide by zero)

Solution dans P2P-MPI :

Le module *check-alive* dans FD va détecter une panne d'application si l'application ne répond pas au message *check-alive*. Après détection de la panne, il arrête de fonctionner (d'envoyer son heartbeat). Les autres noeuds peuvent alors détecter la panne comme la panne d'un noeud.

Expériments

Système :

- Grid5000
- 32 nœuds à grillon (Nancy)
- 32 nœuds à parasol (Rennes)
- 32 nœuds à azur (Nice)

Configuration :

- Protocole gossip (double binaire round-robin)
- $T_{gossip} = 1$ seconde

Etape de test :

- Lance une application MPI.
- Tue tous les processus dans un nœud (killall java).
- Mesure le temps écoulé entre la panne et la date où le nœud le sait.

Expériments

Système :

- Grid5000
- 32 nœuds à grillon (Nancy)
- 32 nœuds à parasol (Rennes)
- 32 nœuds à azur (Nice)

Configuration :

- Protocole gossip (double binaire round-robin)
- $T_{gossip} = 1$ seconde

Etape de test :

- Lance une application MPI.
- Tue tous les processus dans un nœud (killall java).
- Mesure le temps écoulé entre la panne et la date où le nœud le sait.

Expériments

Système :

- Grid5000
- 32 nœuds à grillon (Nancy)
- 32 nœuds à parasol (Rennes)
- 32 nœuds à azur (Nice)

Configuration :

- Protocole gossip (double binaire round-robin)
- $T_{gossip} = 1$ seconde

Etape de test :

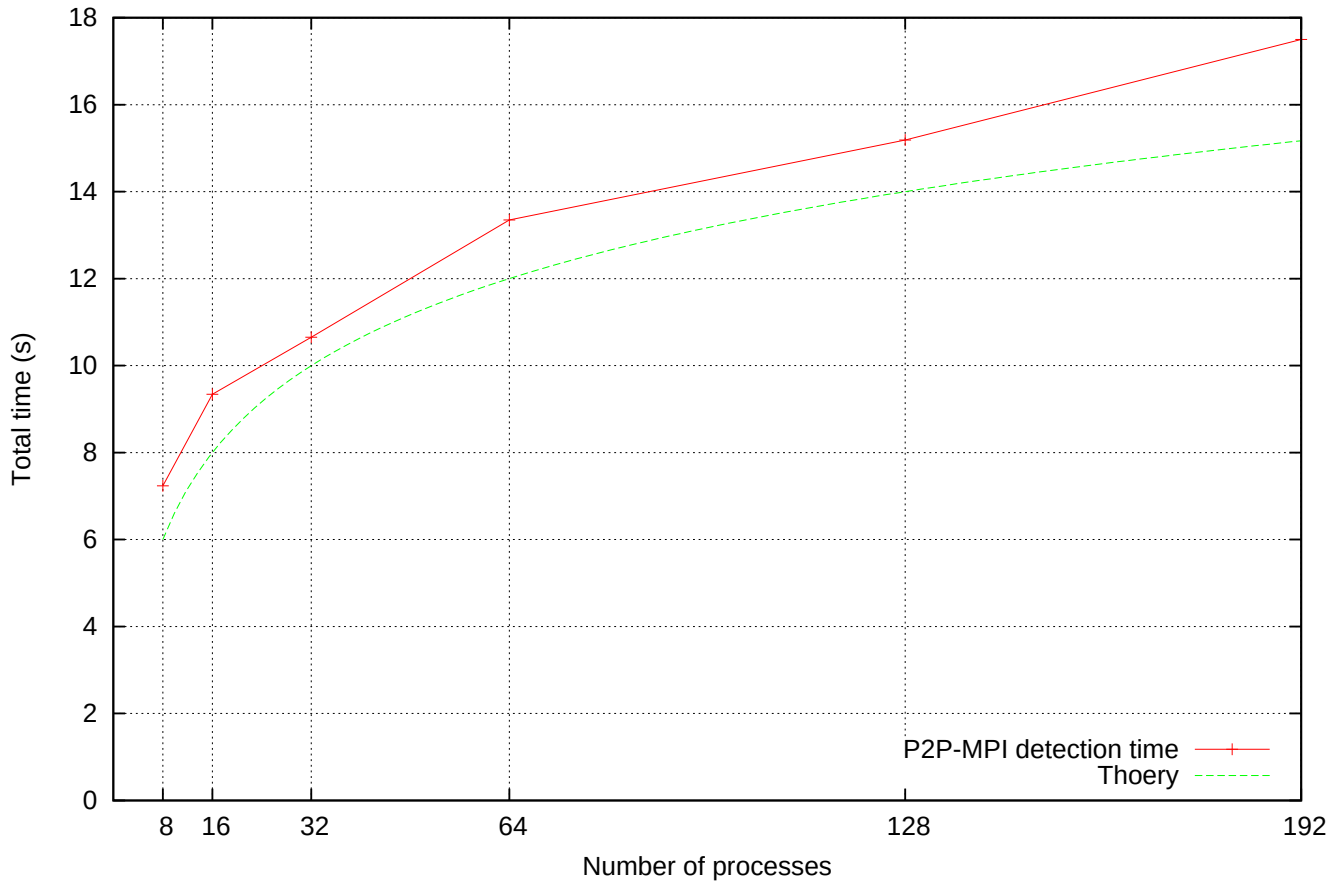
- Lance une application MPI.
- Tue tous les processus dans un nœud (killall java).
- Mesure le temps écoulé entre la panne et la date où le nœud le sait.

Le temps de détection des pannes

Temps de détection :

nodes	Ideal	Min	Max	Avg	Std Deviation
8	6	7.21	7.33	7.29	0.041
16	8	9.16	9.57	9.34	0.144
32	10	10.53	10.84	10.65	0.083
64	12	13.15	13.52	13.35	0.090
128	14	14.83	15.46	15.19	0.154
192	16	16.60	18.17	17.50	0.234

Le temps de détection des pannes



Conclusion

- P2P-MPI propose une exécution **robuste** d'une application parallèle, de manière **transparente**.
- Système de détection de pannes
  - distribué (extensible)
  - déterministe ($2 * log_2(n)$ round)
  - fiable
  - intégré à P2P-MPI (adjustement du heartbeat, check-alive)

Téléchargement

http://grid.u-strasbg.fr/p2pmpi