

# Simulation avancée pour la gestion de ressources des super-ordinateurs

Soutenance de thèse

Adrien Faure

*Supervisé par :*

Olivier Richard<sup>2</sup>, Pascale Rossé-Laurent<sup>1</sup>  
et Denis Trystram<sup>2</sup>



2



1

# Plan de la section 1

- 1 Introduction
- 2 Étude expérimentale des RJMS : Méthodes et état de l'art
  - Extension : Hybridation
- 3 Simulation d'ordonnancement avec consommation de ressources
  - Les profils de jobs
- 4 Validation des *Ptasks*
  - Préparation de l'installation réelle
  - Préparation de la simulation
- 5 Émulation et hybridation avec un RJMS
  - Simunix
  - Batsky
- 6 Conclusion

# Calcul Haute Performances (HPC)

## Calcul de modèles scientifiques

- Nucléaire
- Climat
- Physique



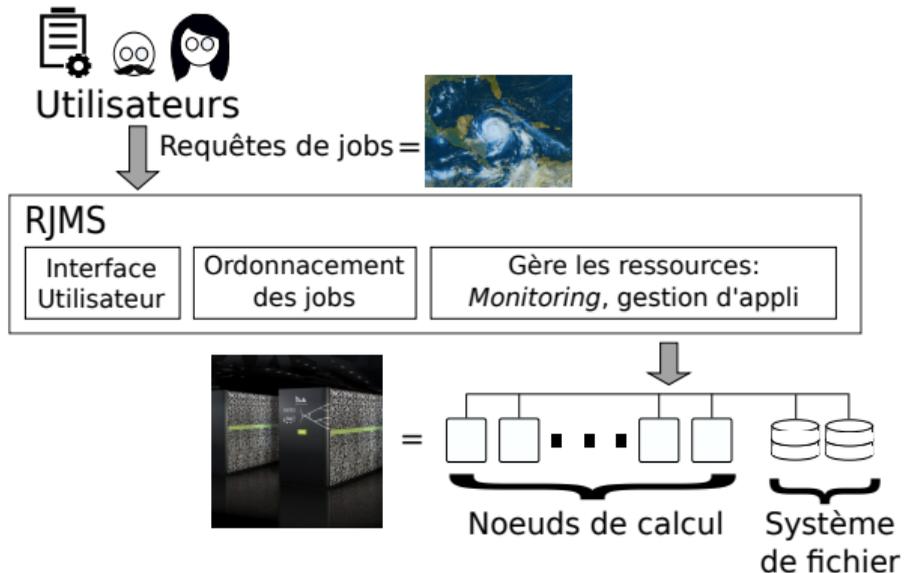
## Organisation en plateforme

- Nœuds de calcul
- Interconnect : Basse latence / Haut débit
- Matériel à la pointe de la technologie



# Gestionnaire de ressources et de jobs

Resources and Jobs Management System (RJMS)



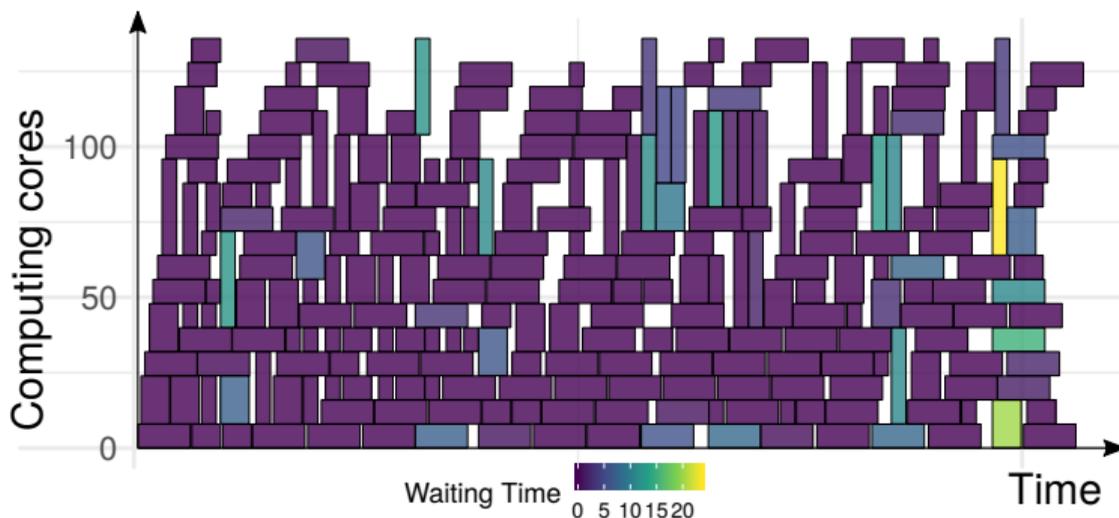
Plateforme partagée → plusieurs applications

RJMS :

- Requêtes utilisateurs = jobs
- Ordonnancement et placement
- Gestion de la plateforme
  - Contrôle l'exécution des jobs
  - *Monitoring* des noeuds
- Logiciel complexe (milliers de lignes de code)

# Exploitation des ressources

Représentation de l'utilisation d'une plateforme du point de vue du RJMS



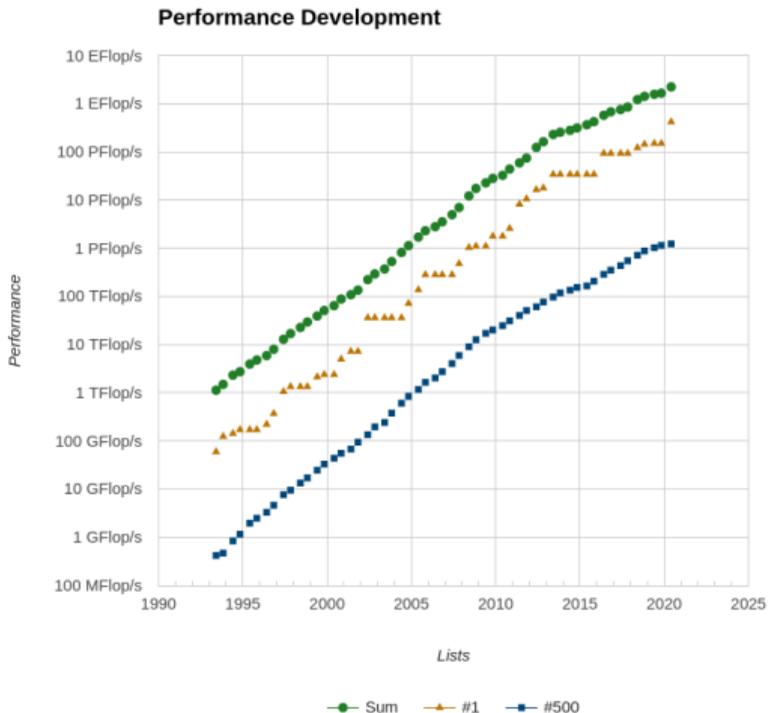
Exemples métriques :

- Minimiser surface vide
- Minimiser les temps d'attentes

Représentation partielle :

- Interférences ?
- Système de fichier ?

# Évolutions et usages des super-calculateurs



## Puissance de calcul :

- Plus de cœurs de calculs
- Ressources hétérogènes :
  - Processeurs spécialisés
  - Systèmes de stockage

## Cas d'utilisation variés :

- Grosse quantité de données
- Anciennes applications
- Apprentissage profond

# Prochaines générations de RJMS

## RJMS d'aujourd'hui

- Ordonnancement centré sur les CPU
  - Gère les cœurs des CPU
  - Ne gère pas les autres ressources
- Les jobs sont des boîtes noires
  - Pas d'info sur leur exécution
  - Chaque job est traité de la même façon

**Les RJMS doivent évoluer pour une meilleure exploitation**

## Évaluation difficile :

- Ordonnancement avec  $\neq$  ressources
- Évaluation des modifications du code
- Changement de plateforme
- Ajouts de nœuds

## Problématique

Comment suivre et évaluer les évolutions des RJMSs ?

# Plan

- 1 Introduction
- 2 Étude expérimentale des RJMS : Méthodes et état de l'art
  - Extension : Hybridation
- 3 Simulation d'ordonnancement avec consommation de ressources
  - Les profils de jobs
- 4 Validation des *Ptasks*
  - Préparation de l'installation réelle
  - Préparation de la simulation
- 5 Émulation et hybridation avec un RJMS
  - Simunix
  - Batsky
- 6 Conclusion

# Plan de la section 2

- 1 Introduction
- 2 Étude expérimentale des RJMS : Méthodes et état de l'art
  - Extension : Hybridation
- 3 Simulation d'ordonnancement avec consommation de ressources
  - Les profils de jobs
- 4 Validation des *Ptasks*
  - Préparation de l'installation réelle
  - Préparation de la simulation
- 5 Émulation et hybridation avec un RJMS
  - Simunix
  - Batsky
- 6 Conclusion

# Expérimenter autours des RJMS ?

## Ressources dédiées ?

- Très couteux
- Pas (forcément) pertinent...

## Besoins de méthodes et outils...

# Classification des expériences

		Application	
		Réalité	Modèle
Plateforme	Réalité	<i>in-vivo</i>	<i>benchmark</i>
	Modèle	<i>in-vitro</i> Émulation	<i>in-silico</i> Simulation

Jens Gustedt et al. **Experimental Methodologies for Large-Scale Systems: a Survey.** [GJQ09]

# Extension de la classification : Hybridation

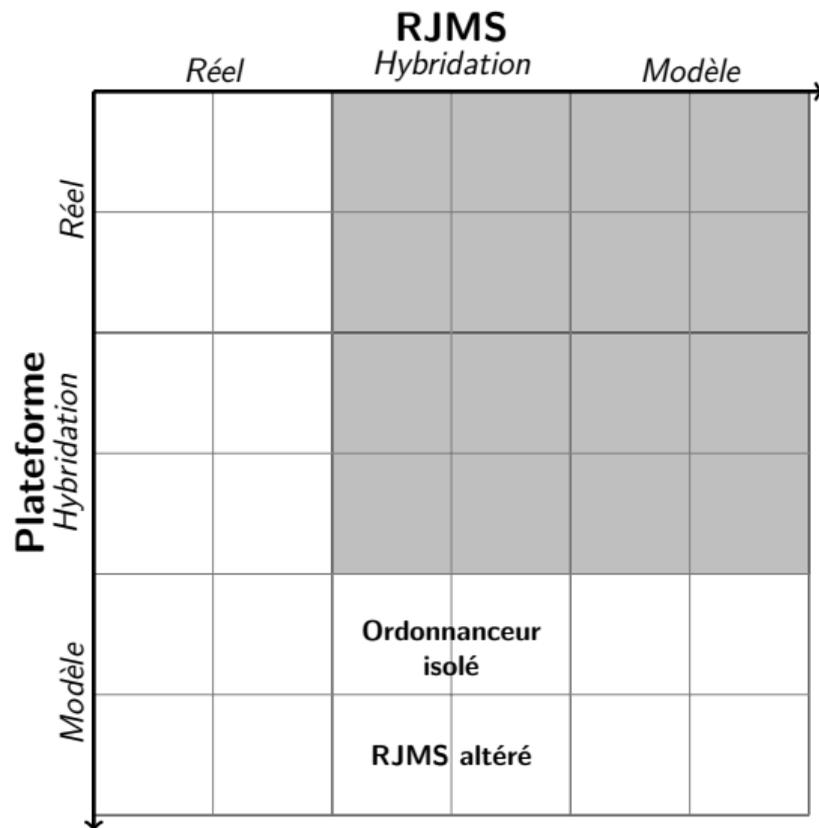
## Deux approches hybrides :

### Ordonnanceur isolé [Dut+16] :

- Ordonnanceur réel isolé → Émulation
- Simulateur de plateforme → Simulation

### RJMS altéré [JDC18] :

- Modification du code → Simulation
- Simulation à partir d'un RJMS → Émulation



# Extension de la classification : Hybridation

## *in-vivo*:

- Testbed

## Émulation :

- Simulateur interne

## Simulation :

- Simulateur à événements discrets

## Hybridation (autre exemples)

### Plateforme altérée :

- Injections de pannes
- Dégradation de performances

### Plateforme virtuelle :

- Machines virtuelles / containers

		RJMS			
		Réel	Hybridation		Modèle
Plateforme Hybridation	Réel	Infrastructure Testbed			
		Plateforme Altérée			
		Plateforme Virtualisée			
Modèle			Ordonnanceur isolé		
		Simulation interne	RJMS altéré	Discrete Event Simulator	

# Plan de la section 3

- 1 Introduction
- 2 Étude expérimentale des RJMS : Méthodes et état de l'art
  - Extension : Hybridation
- 3 Simulation d'ordonnancement avec consommation de ressources
  - Les profils de jobs
- 4 Validation des *Ptasks*
  - Préparation de l'installation réelle
  - Préparation de la simulation
- 5 Émulation et hybridation avec un RJMS
  - Simunix
  - Batsky
- 6 Conclusion

# Évaluation des RJMSs via simulation

## Simuler l'ordonnement :

- 1 Injection d'une charge de travail (liste de jobs)
- 2 Simulation de l'ordonnement
- 3 Métriques

# Évaluation des RJMSs via simulation

## Simuler l'ordonnement :

- 1 Injection d'une charge de travail (liste de jobs)
- 2 Simulation de l'ordonnement
- 3 Métriques

## Exécution des jobs dépend :

- Performances de la plateforme
- Type de job (BigData, HPC, etc.)
- Placement sur la plateforme
- Autres jobs (interférences, contentions etc.)

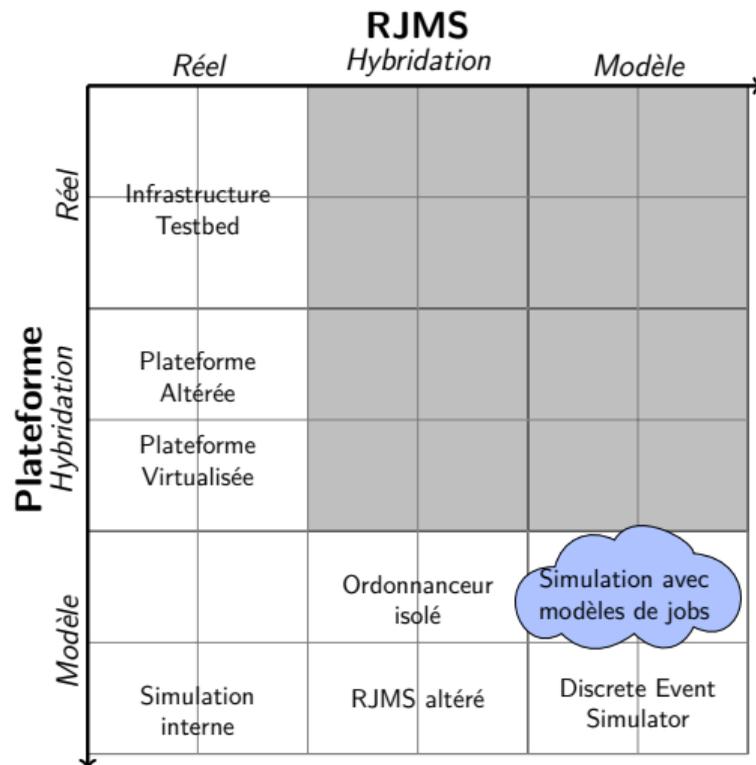
## Simulation avec ces effets ?

# Consommation de ressources des jobs

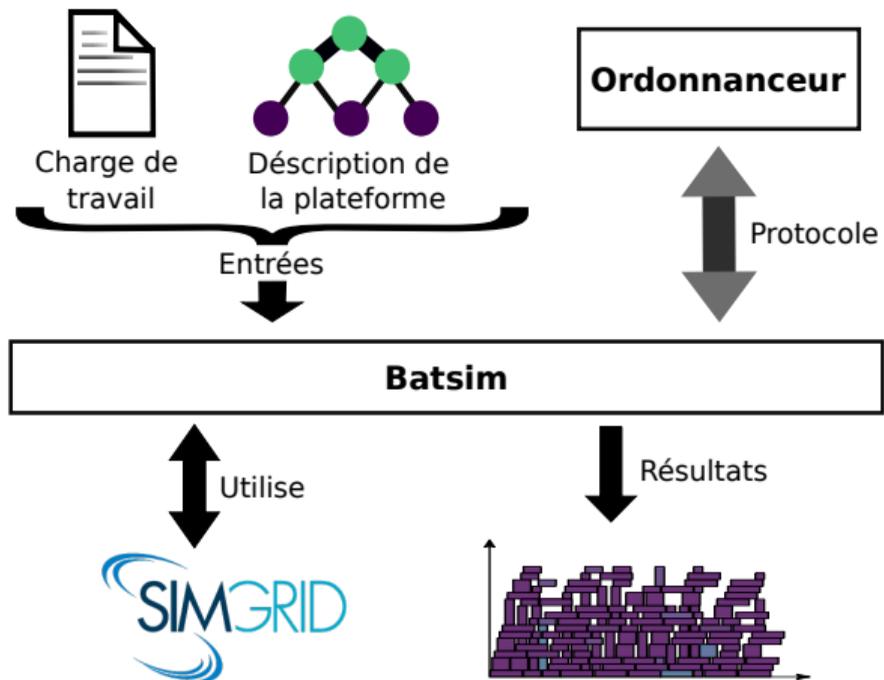
**Consommation de ressources :**  
Simuler le comportement des jobs

- Ralentissements (réseau, IO, CPU etc)
- Interférences inter & intra jobs
- Impact de la topologie réseau
- Énergie consommée
- Indépendant de l'implémentation

**Besoin d'un simulateur**



# Batsim : Simulateur de RJMS et d'infrastructure

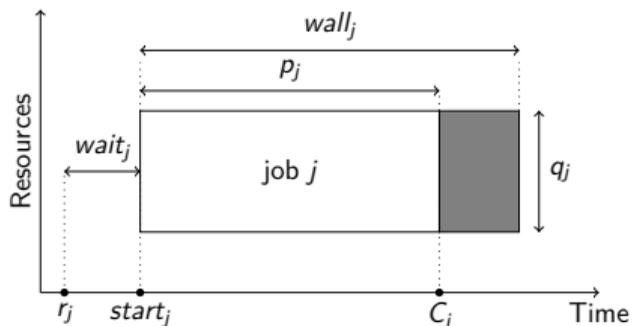


## Points clé

- Séparation RJMS / ordonnancement
- Basé sur SimGrid
- Différents modèles de jobs

Millian Poquet. *Approche par la simulation pour la gestion de ressources.* [Poq17]

# Charge de travail de la simulation (Workload)

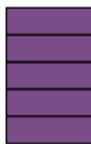


## Liste des jobs

- Date de soumission :  $r_j$
- Nombre de ressources :  $q_j$
- Walltime (spécifique au HPC) :  $wall_j$
- **1 profil de simulation** :  $C_j$  ??

Temps d'exécution → Dépend du profil

## Jobs:



## Profils:

```
{  
  "name": "cpu1",  
  "type": "compute",  
  "flops": [1e3, 1e3],  
  "nb_res" : 2,  
}
```

```
{  
  "name": "cpu2",  
  "type": "compute",  
  "flops": [1e3, ..., 1e3]  
  "nb_res": 5,  
}
```

# Profil délai

## Délai

- Temps d'exécution fixe
- Fourni en avance
- Le plus utilisé

## Entrée du modèle

- Nombre de secondes

## Propriétés

- + Très rapide
- - Aucun effet intra / inter jobs
- Réaliste quand :
  - plateforme homogène
  - pas d'effet de contention

# Profil TiT

## Time Independant Trace (TiT)

- Modèle fin
- Rejeu de trace MPI
- Temps d'exécution dépend de :
  - quantité de travail du job
  - capacité de la plateforme
  - placement
  - autres jobs

## Entrée du modèle

- Trace TiT : Liste d'appels de routines MPI

## Propriétés

- Réservé aux jobs MPI
- + Effets intra / inter jobs
- + Bénéficie de validation [Des+11]
- – Lent (pour l'étude d'une application)
- Réaliste quand :
  - Jobs MPI *statiques*
  - Performances indépendantes des données

# Profile *ptask*

## Modèle de tâche parallèle

- Haut niveau
- Progression uniforme
- Temps d'exécution dépend de :
  - quantité de travail du job
  - capacité de la plateforme
  - placement
  - autres jobs

## Entrées du modèle

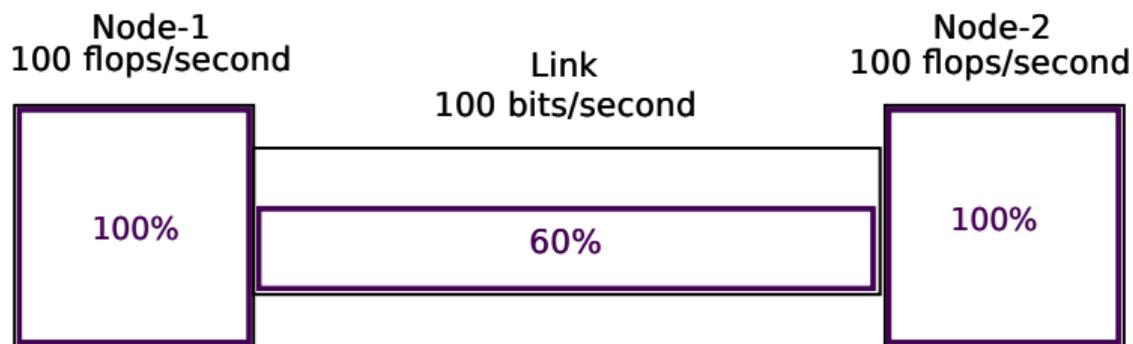
- Utilise plusieurs ressources
  - vecteur de calcul
  - matrice de communication

## Propriétés

- + Effets intra / inter jobs
- + Rapide
- - Pas évalué avant cette thèse
- Réaliste quand : ?

# Progression des *Ptasks*

- $t = 0$  s
- 1 *ptask* (violette)

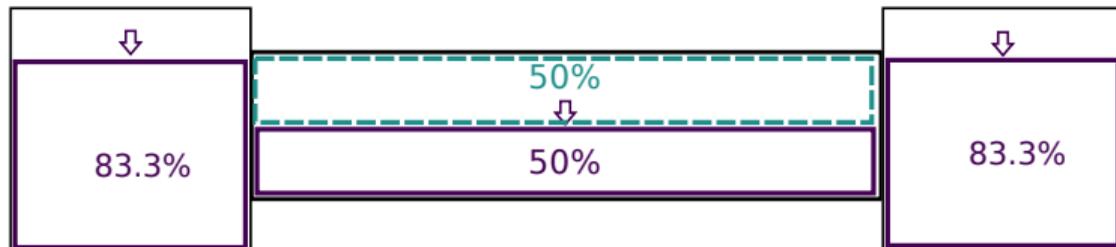


Details of remaining work ( $t = 0$ s)

work ptask	Node-1	Link	Node-2
	1000 flops	600 bits	1000 flops

# Progression des *Ptasks*

- $t = 1$  s
- 1 *ptask*
- 2nd *ptask* bleu



Details of remaining work ( $t = 1$  s)

work ptask	Node-1	Link	Node-2
	900 flops	540 bits	900 flops
	0 flops	1000 bits	0 flops

# Progression des *Ptasks*

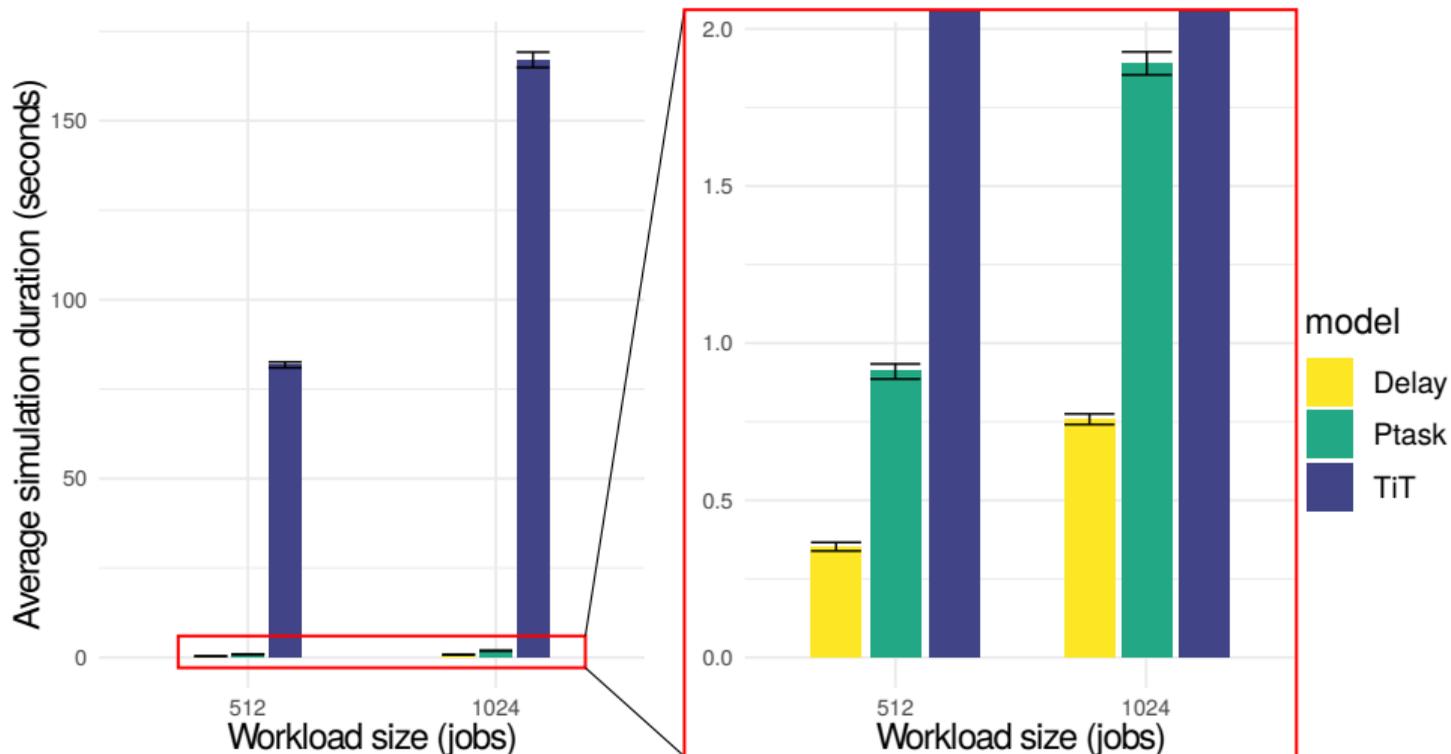


- $t = 6$  s
- 1ère *ptask* violette
- 2ème *ptask* bleu
- 3ème *ptask* verte

Details of remaining work ( $t = 6$  s)

work ptask	Node-1	Link	Node-2
	483 flops	290 bits	483 flops
	0 flops	750 bits	0 flops
	300 flops	0 bits	1000 flops

# Performances des modèles



Pour re-jouer l'expérience: [https://gitlab.inria.fr/adfaure/ptask\\_tit\\_eval](https://gitlab.inria.fr/adfaure/ptask_tit_eval)

# Quel modèle choisir ?

## Inclure la consommation de ressources en simulation?

### Profil Delay

- Très rapide
- Pas d'effet inter/intra
- Peu d'information

### profil *ptask*

- Rapide
- Effets inter / intra
- **Consommation de ressources**
  - Calculs et communications

### profil TiT

- Lent
- Effets inter / intra
- Uniquement pour MPI

## Besoin de validation du modèle

# Quel modèle choisir ?

## Inclure la consommation de ressources en simulation?

### Profil Delay

- Très rapide
- Pas d'effet inter/intra
- Peu d'information

### profil *ptask*

- Rapide
- Effets inter / intra
- **Consommation de ressources**
  - Calculs et communications

### profil TiT

- Lent
- Effets inter / intra
- Uniquement pour MPI

## Besoin de validation du modèle

# Plan de la section 4

- 1 Introduction
- 2 Étude expérimentale des RJMS : Méthodes et état de l'art
  - Extension : Hybridation
- 3 Simulation d'ordonnancement avec consommation de ressources
  - Les profils de jobs
- 4 Validation des *Ptasks***
  - Préparation de l'installation réelle
  - Préparation de la simulation
- 5 Émulation et hybridation avec un RJMS
  - Simunix
  - Batsky
- 6 Conclusion

# Protocole expérimental

- Validation du modèle d'interférences réseau
- Comparaison Simulation / Réalité

## Réalité

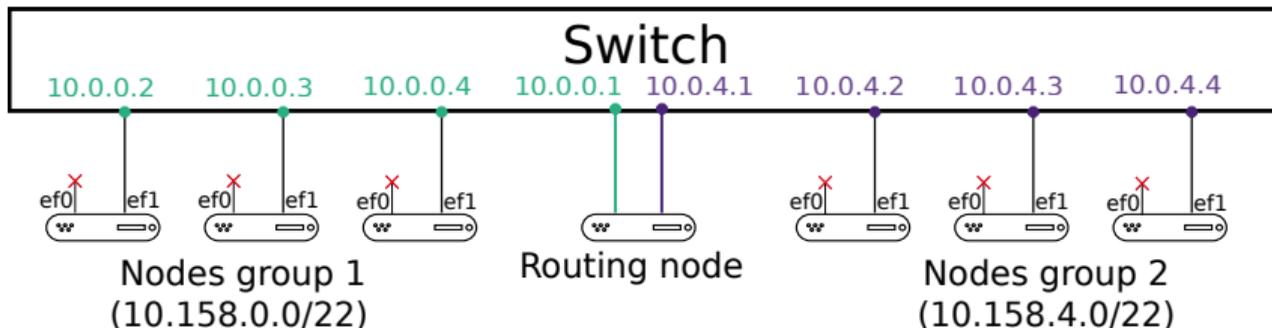
- Application parallèle (MPI)
- Multiplication de matrices
- Générations d'interférences contrôlés

## Simulation

- Exécution d'une *ptask*
- *ptask* correspondante
- Simulation des interférences

# Configuration la plateforme

## Réseau initial



## Pas de réseau (facilement) saturable

### Création d'un point de contention

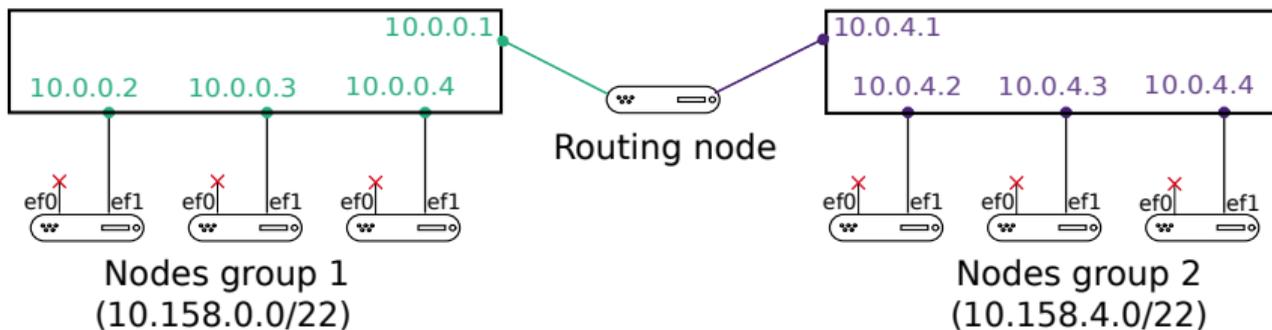
- Deux groupes de machines
- Réparties dans deux sous réseaux
- Communication inter groupe utilise un nœud routeur

### Plateformes Grid'5000 :

- **Grisou** et **Paravance**
- Mêmes nœuds - *Dell poweredge R640*
- Différents switchs

# Configuration la plateforme

## Réseau configuré



## Pas de réseau (facilement) saturable

### Création d'un point de contention

- Deux groupes de machines
- Réparties dans deux sous réseaux
- Communication inter groupe utilise un nœud routeur

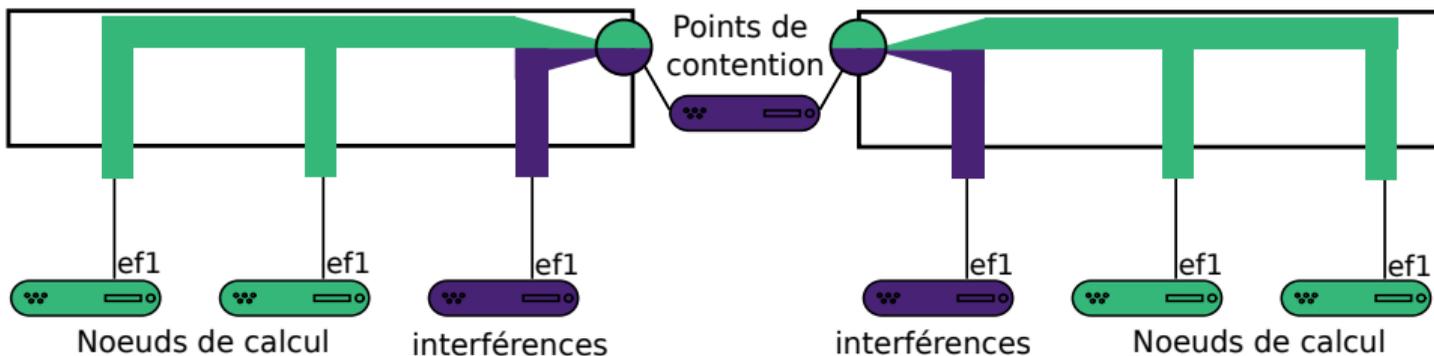
## Plateformes Grid'5000 :

- Grisou et Paravance
- Mêmes nœuds - *Dell poweredge R640*
- Différents switches

# Exécutions : Application et interférences

## Application réelle

- Application homogène
  - Cycles : Communications + Calculs
- Matrices distribuées sur plusieurs nœuds
- 8 nœuds par groupe (16 cœurs / nœud)
- Différents paramétrages
  - Découpages des matrices
  - Broadcast synchrone ou asynchrone



## Interférences

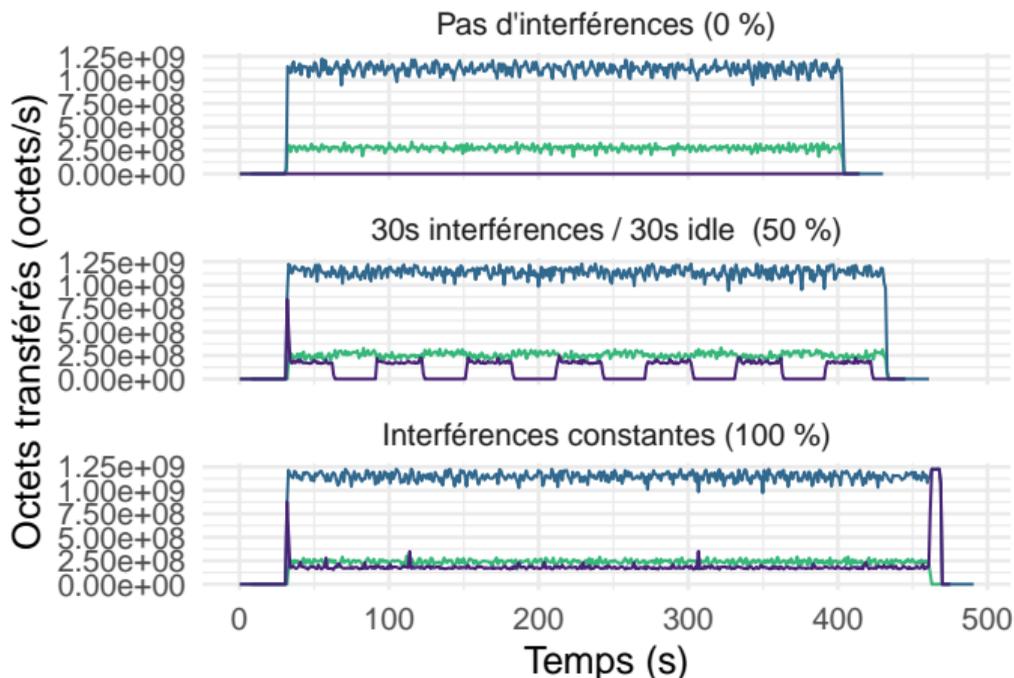
- Tcpkali = logiciel open source
- Génération d'interférences contrôlées
- Période de base 60 s
- % d'interférences : (0 %, 25 %, .. 100 %)
  - 25 % : 15 s interférences / 45 s idle

# Paramétrage et comportement de l'application

- Réseau homogène
- 0 % interférences = plus rapide
- 100 % interférences = plus lent

## Paramétrage de l'appli :

- 50 sous-matrices
- Broadcasts asynchrones



src — MPI (only one host) — Router (eno2) — Tcphkali

# Génération de la *ptask* : Consommation de ressources par l'application

**Besoins pour la simulation :**  
quantité de communications et de calculs

## Vecteur de calculs

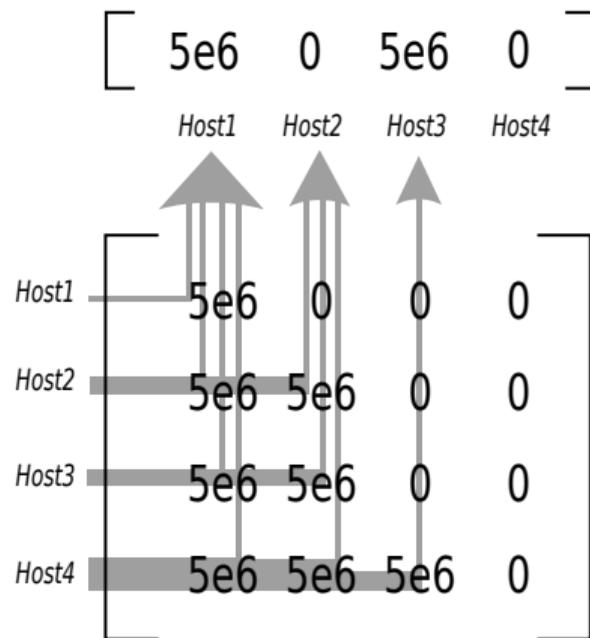
- Extraction depuis l'algorithme
- Dépend de la taille des matrices :

$$\left( \frac{\text{taille\_matrice}}{\sqrt{\text{processus}}} \right)^3$$

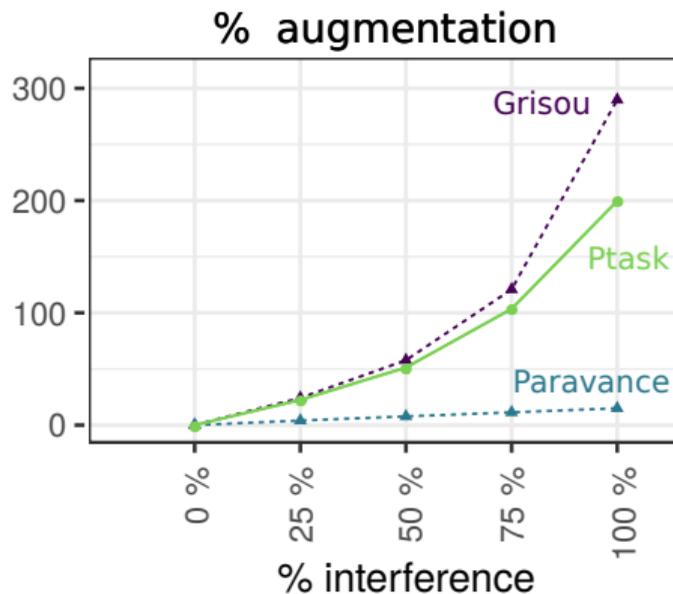
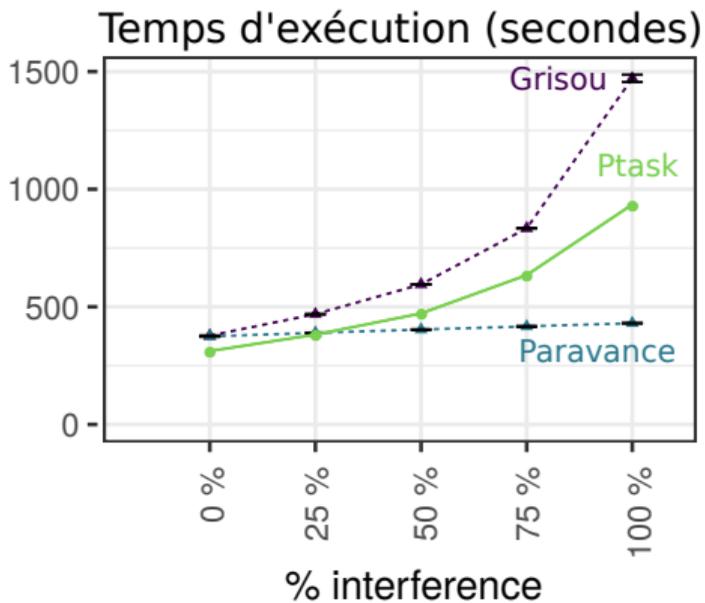
## Matrice de communications

**Dépend de l'algorithme de broadcast**

- bout à bout  $\neq$  point à point
- SimGrid et SMPI  $\rightarrow$  Tracing point à point



# Résultats : Réalité versus Simulation



Temps d'exécution différents → **avec conservation des tendances**

# Cadence de progression de l'application

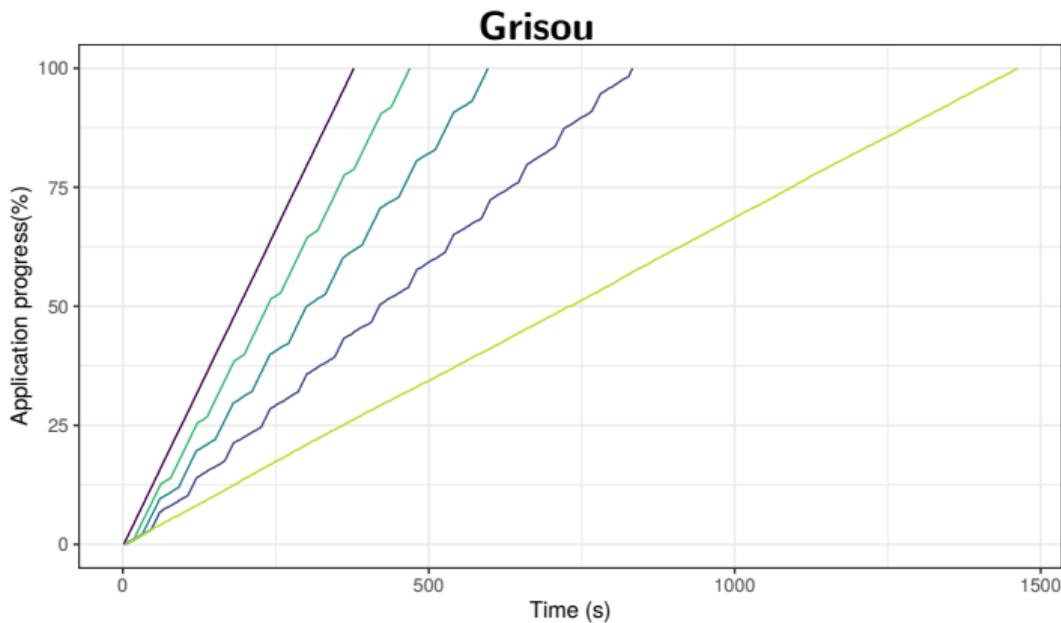
## Monitoring interne

- 0 % interférences
- 25 % interférences
- 50 % interférences
- 75 % interférences
- 100 % interférences

## Progression homogène

2 cadences distinctes :

- rapide sans interférences
- lente avec interférences

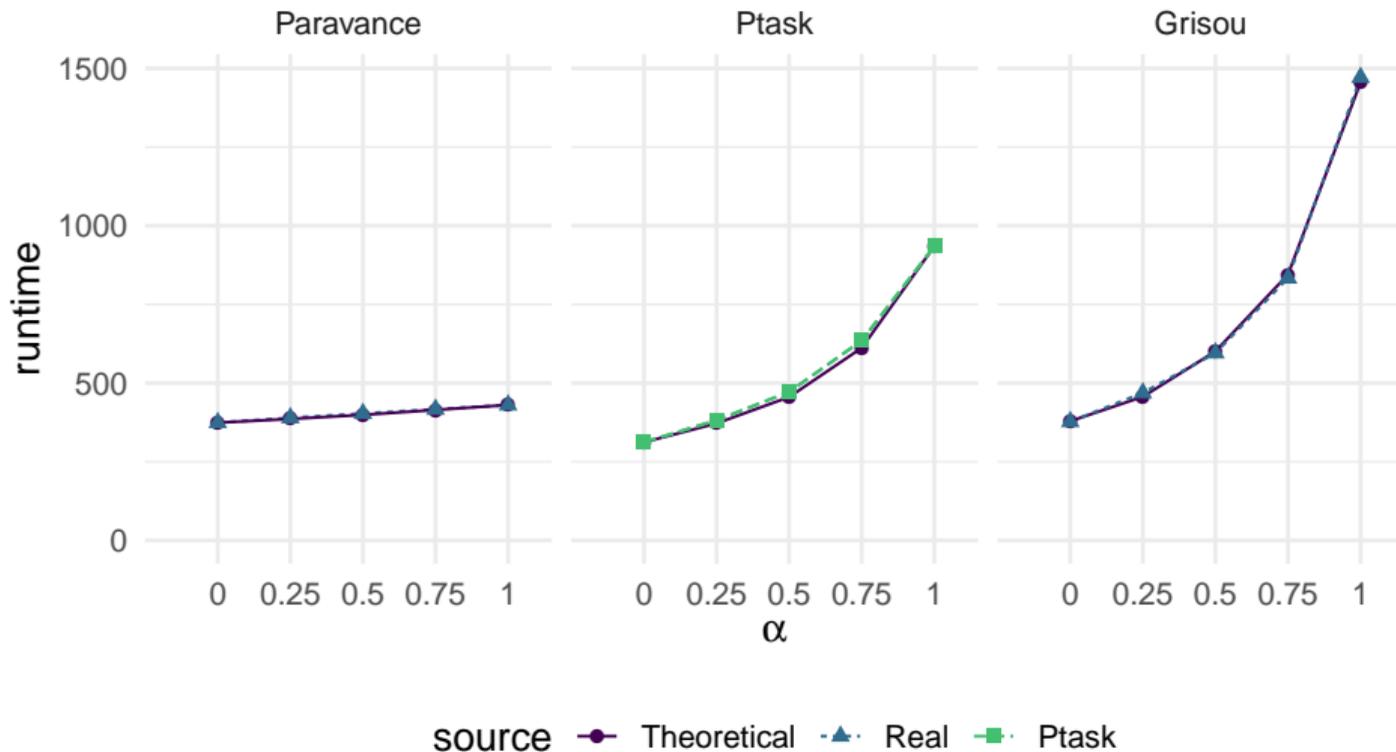


# Un modèle pour les *liar*

$$T((1 - \alpha)C_{rapide} + \alpha C_{lent})$$

- $T =$  Période (60 s)
- $\alpha =$  % interférences
- $C_{rapide}$  coefficient directeur **sans** interférences
- $C_{lent}$  coefficient directeur **avec** interférences

# Un modèle pour les lier



# Conclusion sur les *Ptasks*

## 1er pas de la validation

- Modèle interférences sensé
- Prédiction correcte sans interférences
- ... Adapté pour une routine algébrique

## Évolution nécessaire

Dégradation *ptask*  $\neq$  Dégradation réelle  
Deux plateformes : 2 résultats

### Comment les calibrer ?

- *Monitorer* la simulation ?
- Ajout de paramètres ?

## Prochaines étapes

- Interférences  $\rightarrow$  Application(s)
- Valide pour d'autres applications ?

# Plan de la section 5

- 1 Introduction
- 2 Étude expérimentale des RJMS : Méthodes et état de l'art
  - Extension : Hybridation
- 3 Simulation d'ordonnancement avec consommation de ressources
  - Les profils de jobs
- 4 Validation des *Ptasks*
  - Préparation de l'installation réelle
  - Préparation de la simulation
- 5 Émulation et hybridation avec un RJMS
  - Simunix
  - Batsky
- 6 Conclusion

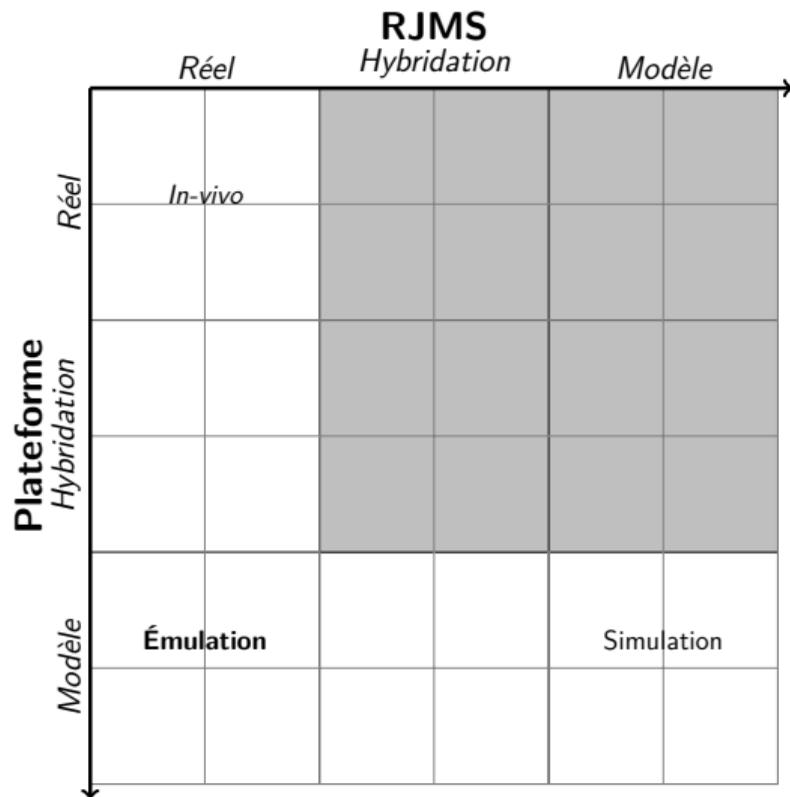
# Comment expérimenter avec un vrai RJMS ?

## Intérêts ?

- Testing
- Scénarios exploratoires
  - Plus de nœuds
  - Différentes topologies réseaux

## Nos options :

- Simulation :
  - Maintenir modèle+implémentation (couteux)
- *In-vivo* (vrai RJMS+plateforme réelle) :
  - Couteux (temps, énergie) et complexe
  - Limité aux plateformes disponibles
- **Émulation** (vrai RJMS+plateforme modélisée):
  - Moins couteux
  - Mais comment ?



# Emulation et hybridation : vrai RJMS avec un modèle de plateforme

## Plateforme altérée :

- Couteux
- Limitée aux plateformes disponibles

## Plateforme virtualisée :

- + de nœuds - de performances
- Couteux en temps (accélération ?)

## Simulation interne :

- Quand disponible
- Modèle de plateforme (trop) simple

## RJMS altérée :

- Modification du code (à maintenir)
- Difficilement extensible

## Ordonnanceur isolé :

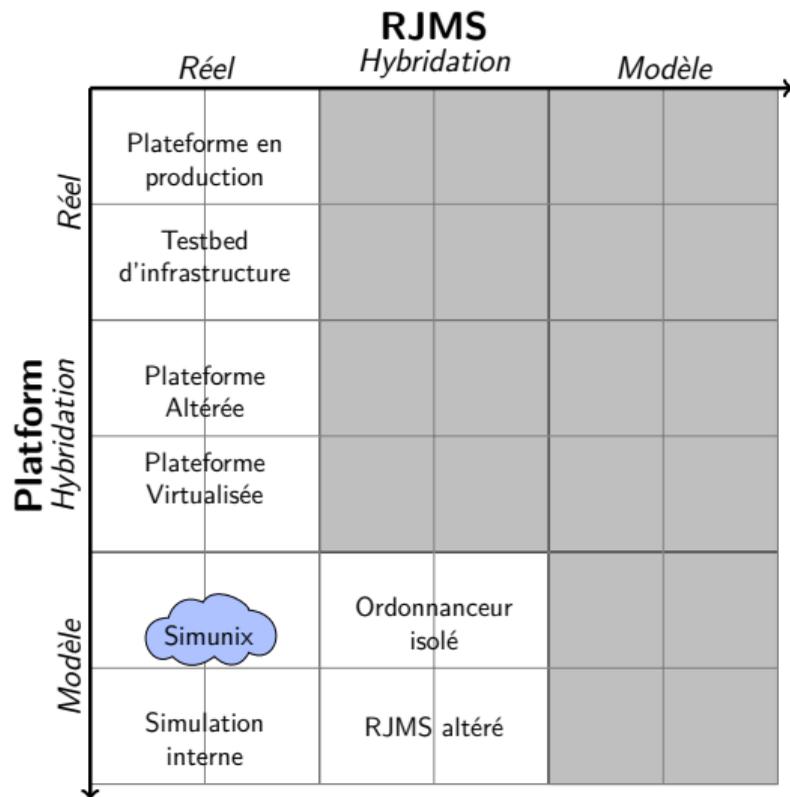
- Difficilement extensible

		RJMS		
		Réel	Hybridation	Modèle
Plateforme	Réel			
	Hybridation	Plateforme Altérée		
Modèle	Hybridation	Plateforme Virtualisée		
	Modèle		Ordonnanceur isolé	
		Simulation interne	RJMS altéré	

# Deux nouvelles approches : Simunix et Batsky

## Simunix : vrai RJMS + SimGrid

- Émulation pure
- Pas de modification du code
- Modèle de plateforme : SimGrid
- Interception de la *libc*



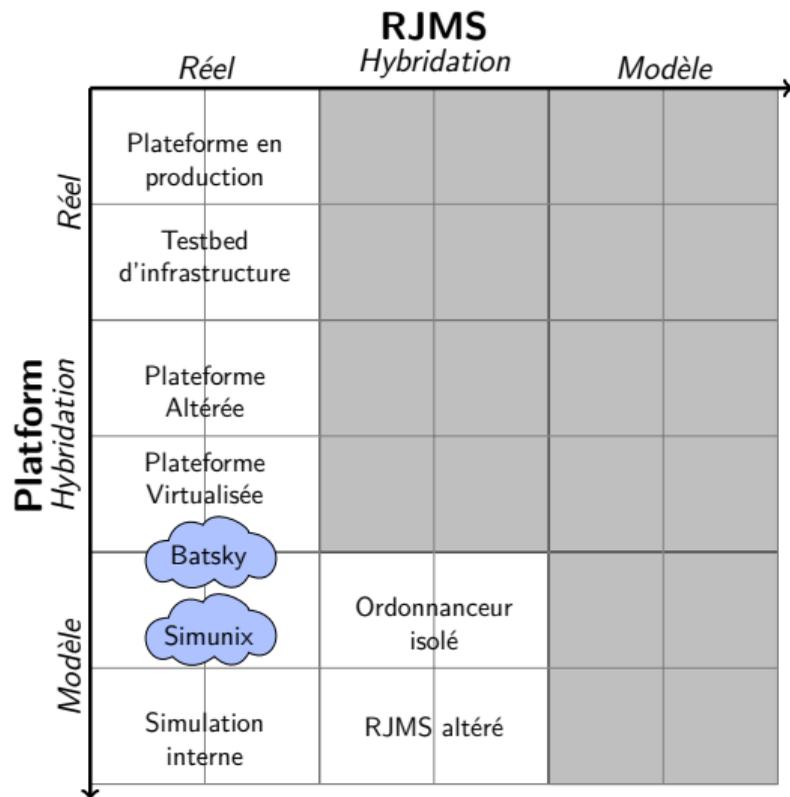
# Deux nouvelles approches : Simunix et Batsky

## Simunix : vrai RJMS + SimGrid

- Émulation pure
- Pas de modification du code
- Modèle de plateforme : SimGrid
- Interception de la *libc*

## Batsky : vrai RJMS + Batsim

- Approche hybride
- Pas de modification du code
- Modèle de plateforme : Batsim
- Interception du temps



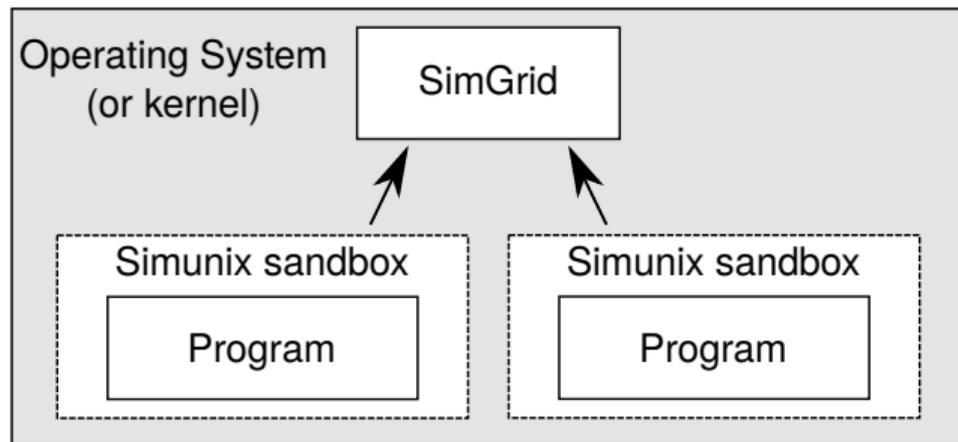
# Principes de Simunix

## Bac à sable :

- Interception de la libc
- Remplacement par SimGrid

## SimGrid simule la plateforme :

- Le réseau
- Le temps
- Threads et processus



# Simunix : exemple avec Slurm

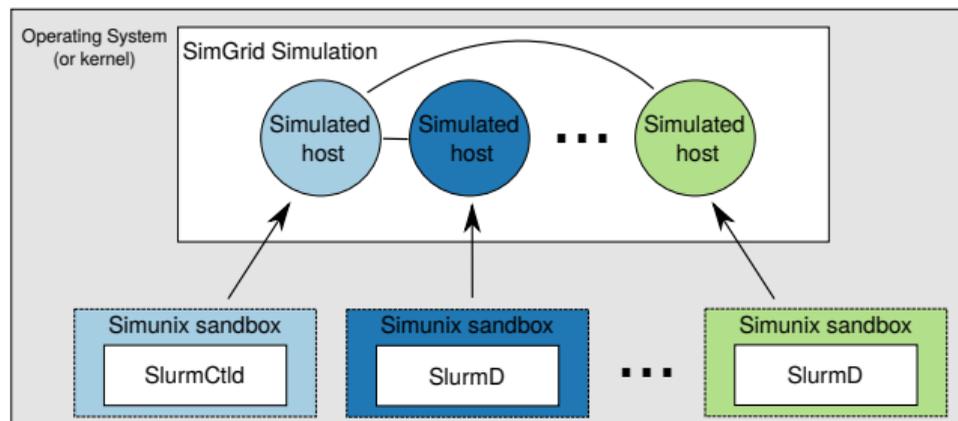
## Slurm = RJMS Open Source

### Installation de Slurm

- Un contrôleur : ***SlurmCtld***
  - Ordonnancement
  - Interface utilisateurs
- Des daemons : ***SlurmDs***
  - Exécute les jobs
  - *Monitoring*

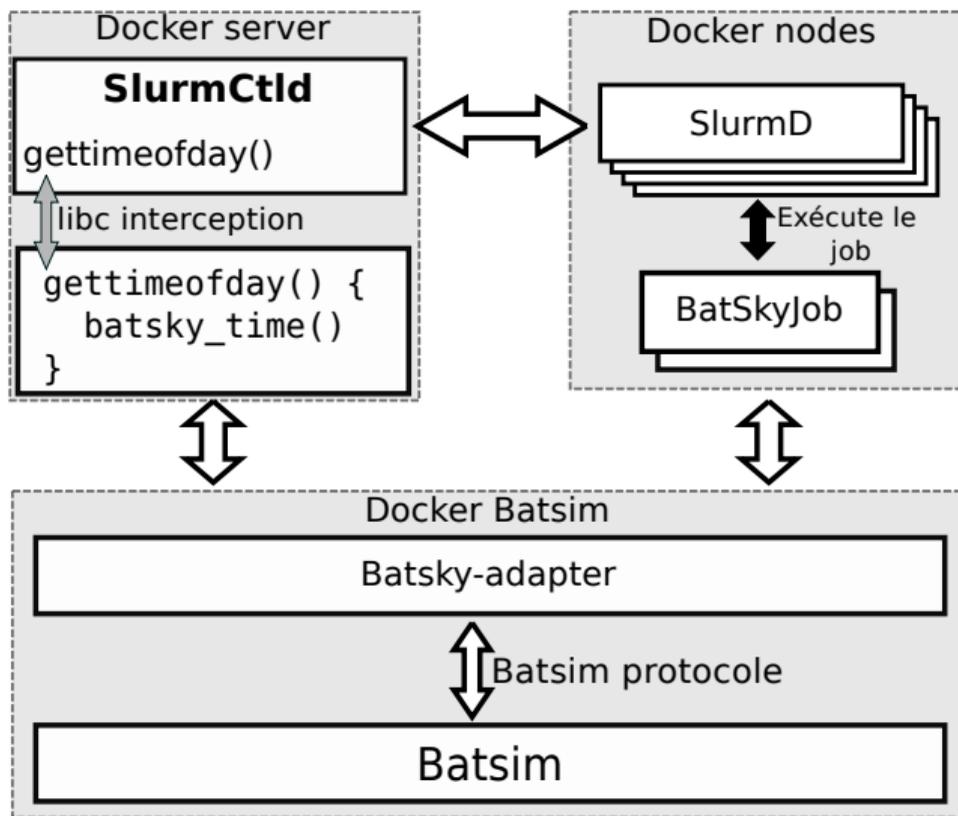
### A fonctionné un moment :

- 10 **SlurmD**
- Différentes versions de Slurm

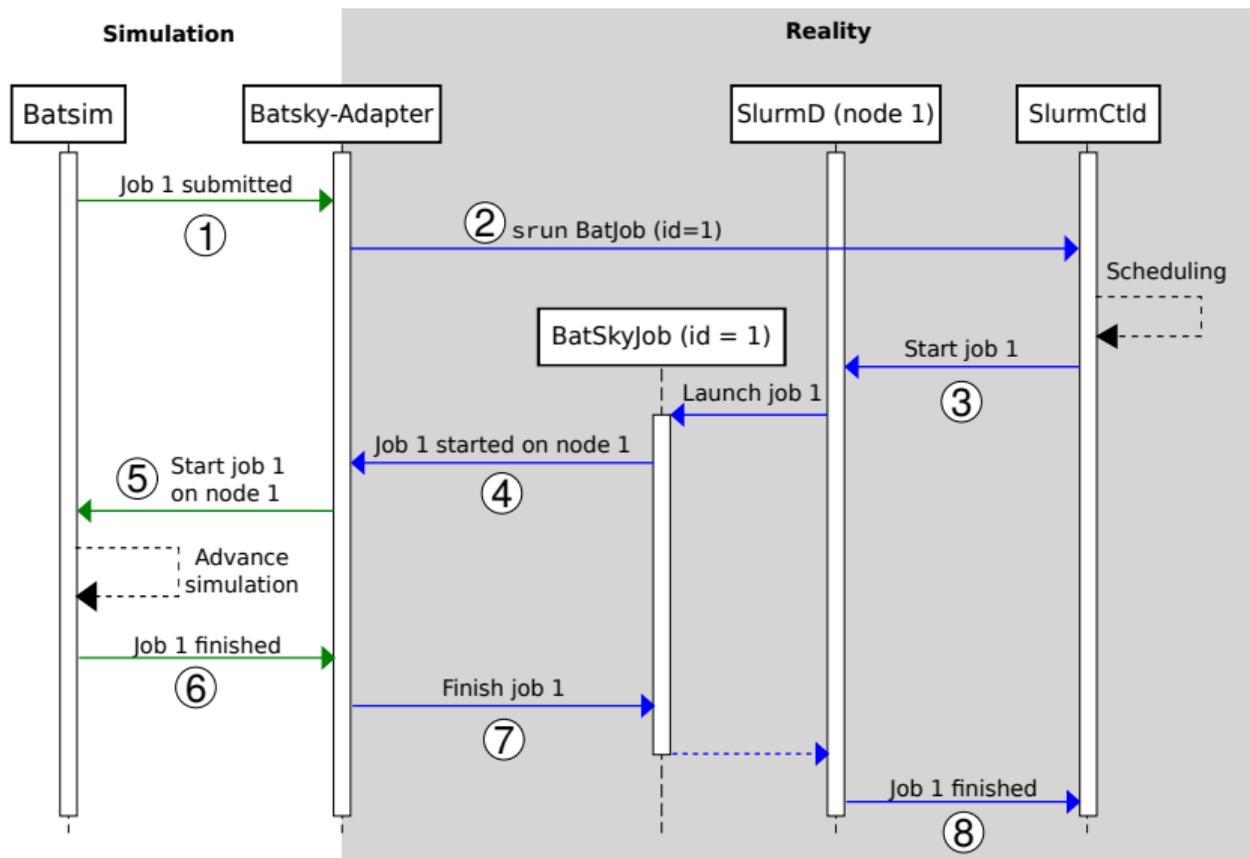


# Batsky

- Interception du temps
- Ordonnanceur Batsim (*Batsky-adapter*)
- **BatSkyJobs** exécutés par le RJMS
  - Envoi d'information à *Batsky-adapter*
  - Gère le temps d'exécution



# Batsky : Diagramme de séquence



# Batsky : rapidité et cohérence de l'exécution

## Accélération

- Interception du temps → sauts temporels

## Certains bons dangereux

- Initialisation
- Lancement d'un job

## Garantir la cohérence

- Instauration de phases d'exécution «normales»
- Temps s'écoule normalement
- Ralentit la simulation

**Temps des phases = compromis cohérence / performances**

# Conclusion sur l'émulation

- Pas de modifications de code
- RJMS sur un ordinateur
- Modèle de plateforme

## Simunix

### Approche complète

- Prototype avec Slurm
- Complexe à mettre en place
- Passage à l'échelle ?

## Batsky

### Approche hybride

- Profite du design des RJMS
- Extensible à d'autres RJMS
- Réservé au RJMS
- Complexe à mettre en place

# Plan de la section 6

- 1 Introduction
- 2 Étude expérimentale des RJMS : Méthodes et état de l'art
  - Extension : Hybridation
- 3 Simulation d'ordonnancement avec consommation de ressources
  - Les profils de jobs
- 4 Validation des *Ptasks*
  - Préparation de l'installation réelle
  - Préparation de la simulation
- 5 Émulation et hybridation avec un RJMS
  - Simunix
  - Batsky
- 6 Conclusion

# Contributions

## Simulation :

- Évaluation d'algorithmes d'ordonnancement par simulation réaliste [FPR18]
- Évaluation du modèle *ptask*

## Émulation et hybridation :

- Contributions sur des techniques d'émulation pour les RJMSs (Batsky, Simunix)

## Ordonnancement :

- Online Scheduling with Redirection for Parallel Jobs [Fau+20]

## Reproductibilité :

- Considering the Development Workflow to Achieve Reproducibility with Variation [MFR18]
- Tutoriel les expériences répétable avec Nix (Seminaire)

# Conclusion et perspectives

## Simulation → Consommation de ressources

- Simulations sensées
- Nouveaux scénarios ouverts à la simulation
  - Appréhender de nouvelles ressources : Énergie, IO, etc
  - Dimensionnement de plateformes

## Émulation hybride → RJMS avec Batsim

- Simunix = Approche générique
- Batsky : taillé pour les RJMS
  - POC avec Slurm et Kubernetes
  - Branchement avec Batsim

## Prochaines étapes

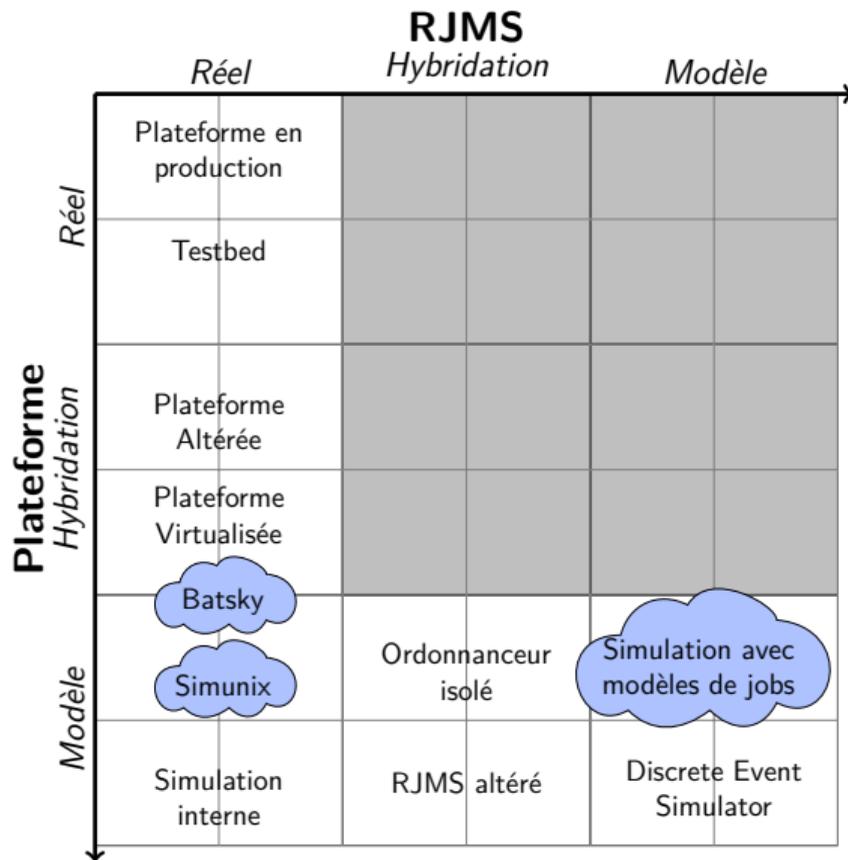
- Portée des *ptasks* ?
  - Quelle applications ?
  - Possible depuis traces de monitoring ?
- Autres ressources : IO [Mer19], énergie etc. ?

## Prochaines étapes

- Évaluation de Batsky :
  - Performance / cohérence
  - Déterminisme ?
- Simunix :
  - Tester d'autres applications : *torrent* ?

Merci !

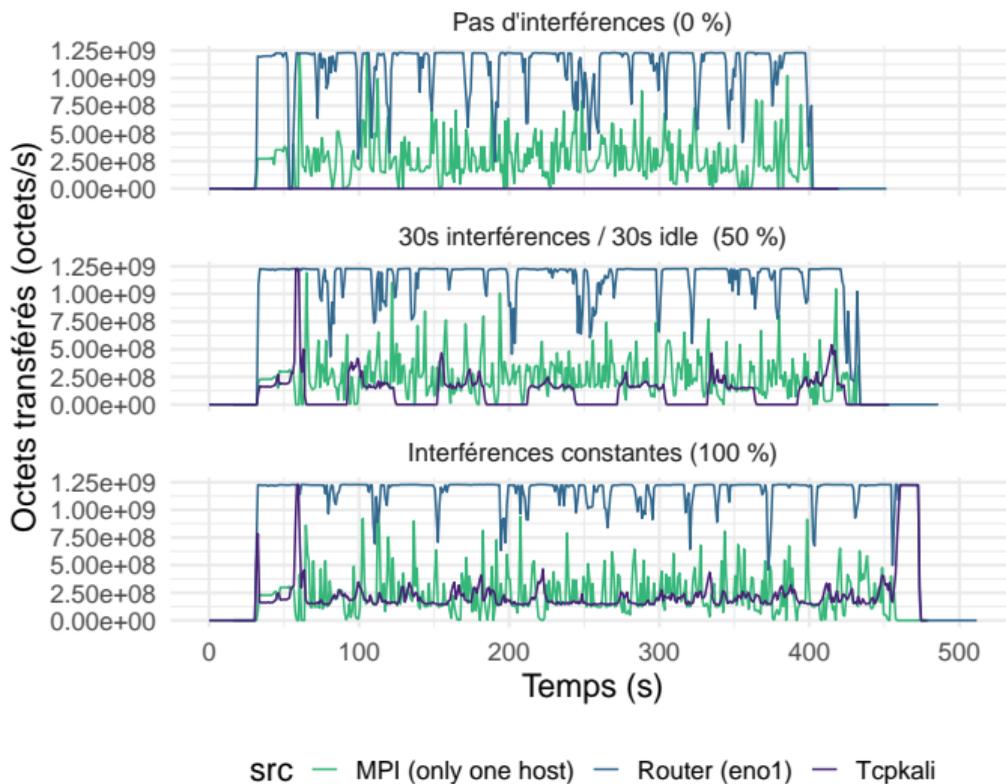
# Hybridation : vue complète



# Comportement de l'application

Plateforme : Paravance Broadcasts asynchrones

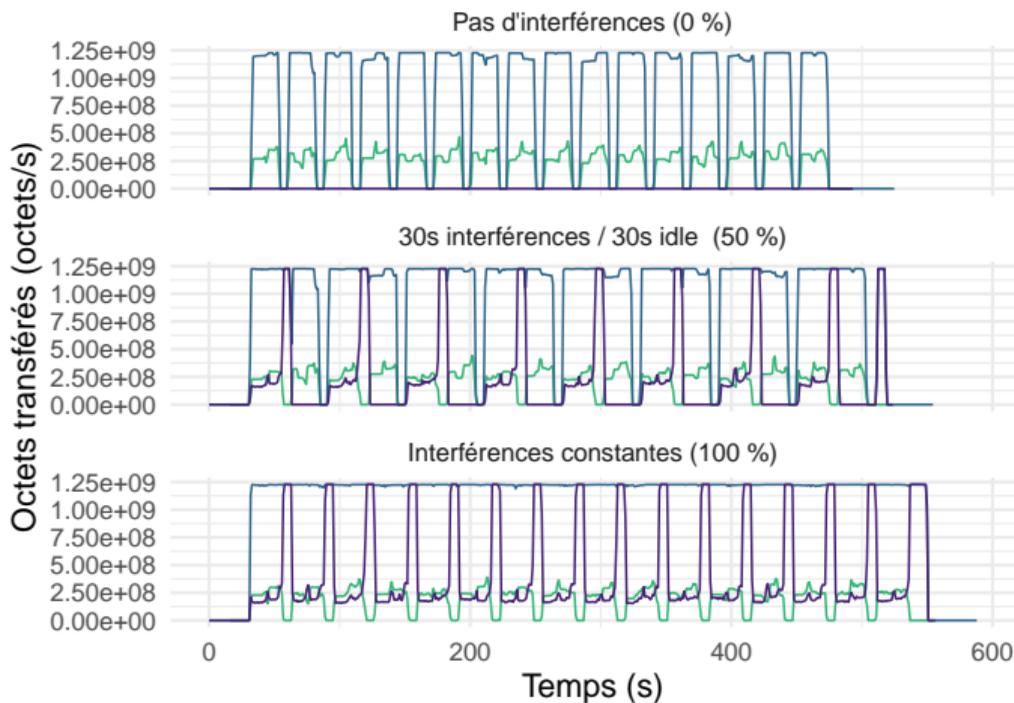
- Communications chaotiques
  - Pas de sous matrice
  - Broadcasts asynchrones



# Comportement de l'application

Plateforme : Paravance Broadcasts synchrones

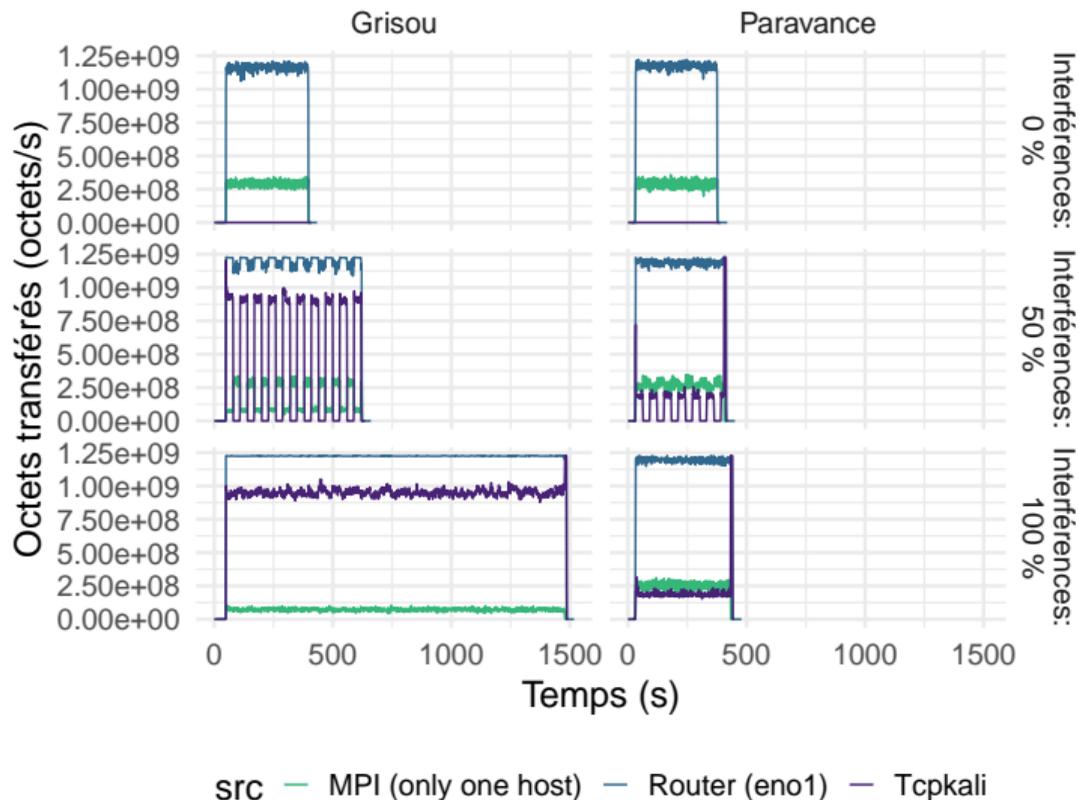
- Communications stables
- **Comportement non-uniforme**
  - Pas de sous matrice
  - Broadcasts synchrones



SRC — MPI (only one host) — Router (eno1) — Tcphali

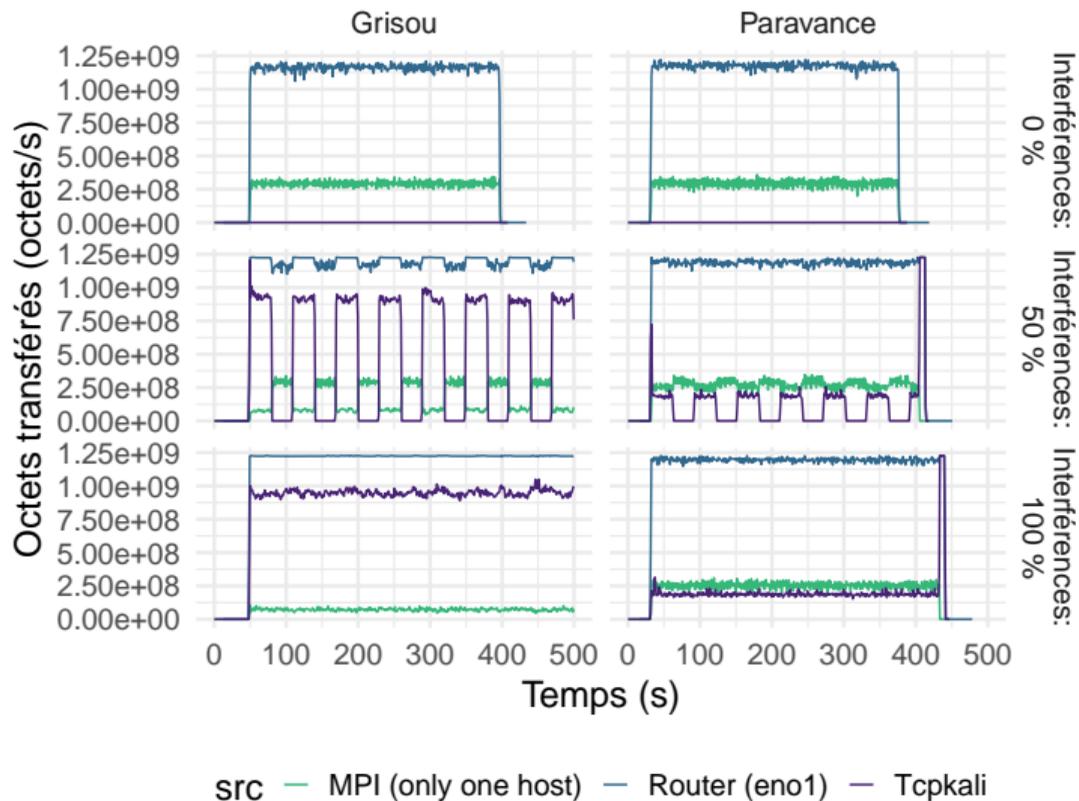
# Différence entre Paravance et Grisou

- Répartition de la charge réseau  $\neq$
- Mêmes noeuds: Dell PowerEdge R630
- Switchs différents:
  - Grisou: *Cisco Nexus 950*
  - Paravance: *Nexus 56128P*



# Différence entre Paravance et Grisou

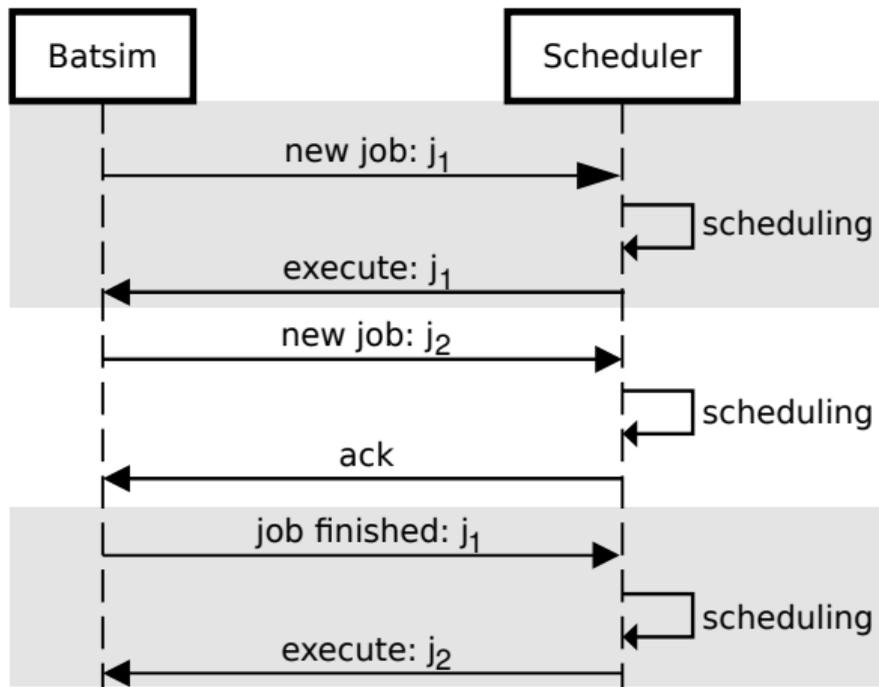
- Répartition de la charge réseau  $\neq$
- Mêmes noeuds: Dell PowerEdge R630
- Switchs différents:
  - Grisou: *Cisco Nexus 950*
  - Paravance: *Nexus 56128P*



# Réalisation : Mécanismes d'interceptions

- ***LD\_PRELOAD*** (Simunix v2)
  - Variable d'environnement
  - Facile à mettre en place
  - Contaminant
- ***ELF hooking*** (Simunix v1)
  - Modification du ELF au runtime
  - Difficile à mettre en place
  - Utilisation des fonctions remplacées
- **Changer la libc** (Batsky)
  - Compilé avec une libc modifiée
  - Difficile à mettre en place
  - Utilisation des fonctions remplacées

# Exemple d'interaction Batsim / ordonnanceur



- Protocole réseaux
- Synchrone

# RemoteSimGrid

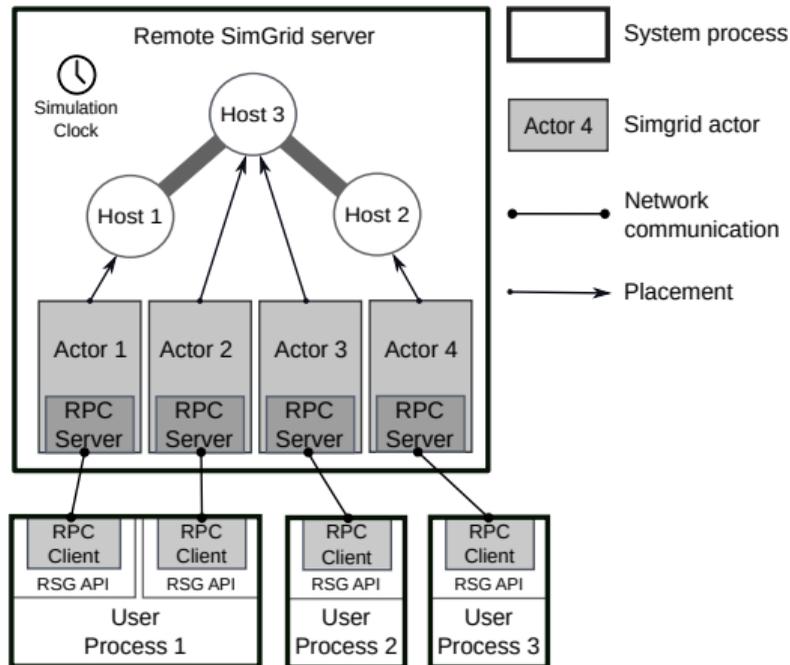
**SG est monolithique** → RJMS =  
Multiprocessus

- Partage des variables globales
- Partage des informations systèmes

## RemoteSimGrid

Isolation des acteurs SimGrid

- Processus systèmes
- Threads



# Interception et Simulation de la librairie C

SG est Monoprocessus  $\neq$  RJMS est Multiprocessus

- Partage des variables globales
- Partage des informations systèmes

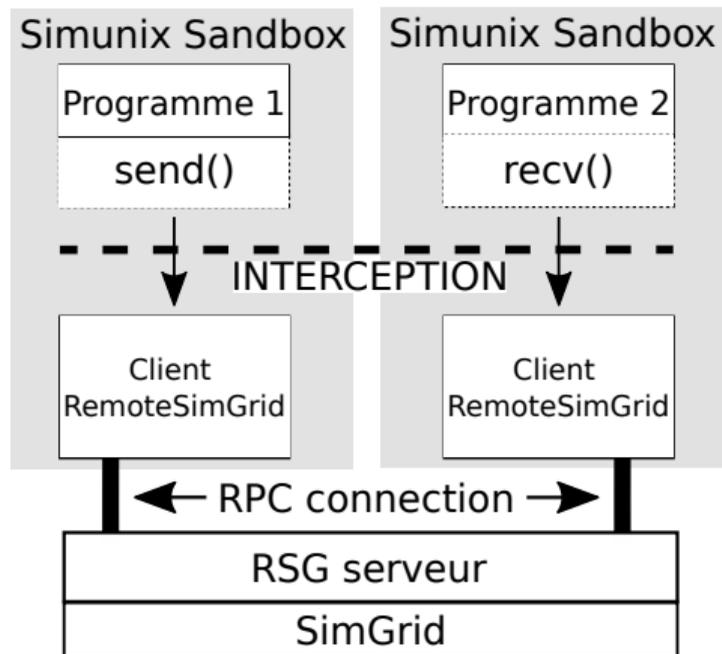
## RemoteSimGrid

Isolation des acteurs SimGrid

- Processus systèmes
- Threads

**Interception** → **Fonction simulée**

- Comportement similaire
- Avec RemoteSimGrid



# Profil Séquence

## Séquence de profils

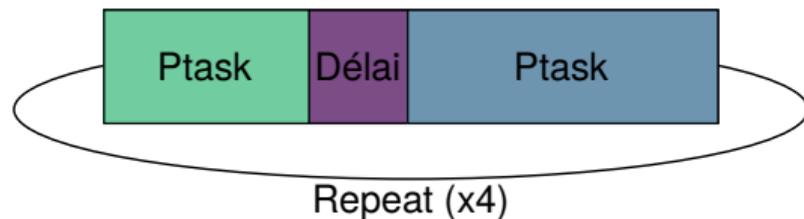
- Composition de modèles
  - Exécutés en séquence
  - Avec un nombre de répétitions

## Entrées du modèle

- Séquence de profils
- Nombre de répétitions

## Propriété

- Hérite des profils qui le compose



## Liens vers les différents projets

- Redirection: Code et expériences <https://gitlab.inria.fr/adfaure/evipar>.
- Évaluation des profiles : [https://gitlab.inria.fr/adfaure/ptask\\_tit\\_eval](https://gitlab.inria.fr/adfaure/ptask_tit_eval)
- Évaluation des *Ptasks* : <https://gitlab.inria.fr/batsim/ptask-eval>
- Batsky : <https://github.com/oar-team/arion-batsky>
- Simunix : <https://framagit.org/simgrid/sgwrap>

# Références I

-  Frederic Desprez et al. “Assessing the Performance of MPI Applications through Time-Independent Trace Replay”. In: *2011 International Conference on Parallel Processing Workshops, ICPPW 2011, Taipei, Taiwan, Sept. 13-16, 2011*. Ed. by Jang-Ping Sheu and Cho-Li Wang. IEEE Computer Society, 2011, pp. 467–476. DOI: 10.1109/ICPPW.2011.33. URL: <https://doi.org/10.1109/ICPPW.2011.33>.
-  Pierre-François Dutot et al. “Batsim: A Realistic Language-Independent Resources and Jobs Management Systems Simulator”. In: *Job Scheduling Strategies for Parallel Processing - 19th and 20th International Workshops, JSSPP 2015, Hyderabad, India, May 26, 2015 and JSSPP 2016, Chicago, IL, USA, May 27, 2016, Revised Selected Papers*. Ed. by Narayan Desai and Walfredo Cirne. Vol. 10353. Lecture Notes in Computer Science. Springer, 2016, pp. 178–197. DOI: 10.1007/978-3-319-61756-5\_10. URL: [https://doi.org/10.1007/978-3-319-61756-5%5C\\_10](https://doi.org/10.1007/978-3-319-61756-5%5C_10).
-  Adrien Faure, Millian Poquet, and Olivier Richard. “Évaluation d’algorithmes d’ordonnancement par simulation réaliste”. working paper or preprint. Apr. 2018. URL: <https://hal.inria.fr/hal-01779936>.

## Références II

-  Adrien Faure et al. “Online Scheduling with Redirection for Parallel Jobs”. In: *2020 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2020, New Orleans, LA, USA, May 18-22, 2020*. IEEE, 2020, pp. 326–329. DOI: 10.1109/IPDPSW50202.2020.00066. URL: <https://doi.org/10.1109/IPDPSW50202.2020.00066>.
-  Jens Gustedt, Emmanuel Jeannot, and Martin Quinson. “Experimental Methodologies for Large-Scale Systems: a Survey”. In: *Parallel Process. Lett.* 19.3 (2009), pp. 399–418. DOI: 10.1142/S0129626409000304. URL: <https://doi.org/10.1142/S0129626409000304>.
-  Ana Jokanovic, Marco D’Amico, and Julita Corbalán. “Evaluating SLURM Simulator with Real-Machine SLURM and Vice Versa”. In: *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems, PMBS@SC 2018, Dallas, TX, USA, November 12, 2018*. IEEE, 2018, pp. 72–82. DOI: 10.1109/PMBS.2018.8641556. URL: <https://doi.org/10.1109/PMBS.2018.8641556>.

## Références III

-  **Michael Mercier.** “Contribution to High Performance Computing and Big Data Infrastructure Convergence”. 2019GREAM031. PhD thesis. 2019. URL: <http://www.theses.fr/2019GREAM031/document>.
-  **Michael Mercier, Adrien Faure, and Olivier Richard.** “Considering the Development Workflow to Achieve Reproducibility with Variation”. In: *SC 2018 - Workshop: ResCuE-HPC*. Dallas, United States, Nov. 2018, pp. 1–5. URL: <https://hal.inria.fr/hal-01891084>.
-  **Millian Poquet.** “Simulation approach for resource management. (Approche par la simulation pour la gestion de ressources)”. PhD thesis. Grenoble Alpes University, France, 2017. URL: <https://tel.archives-ouvertes.fr/tel-01757245>.