Programmation des systèmes parallèles distribués: tolérance aux pannes, résilience et adaptabilité

Samir JAFAR

Directeurs de thèse
Denis TRYSTRAM & *Jean-Louis* ROCH

Projet MOAIS
Laboratoire ID-IMAG (UMR 5132), Grenoble, France.

30 juin 2006

Motivations

Applications Parallèles de longue durée d'exécution

Plates-formes de grande taille (ex : Grid5000)

- Nombre important de ressources
- Environnements dynamiques
- Environnements hétérogènes

$\Rightarrow$ Problèmes de sûreté de fonctionnement

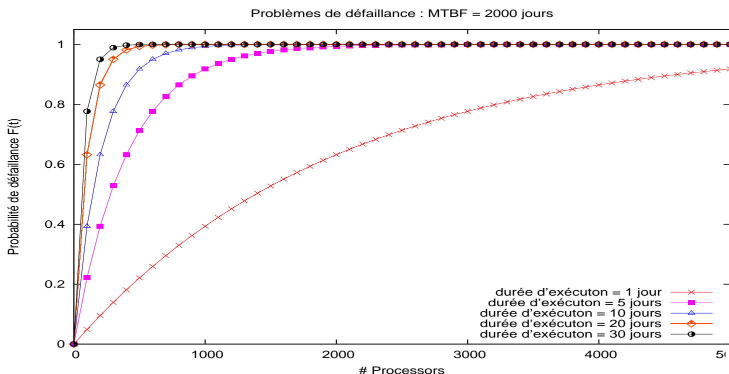
- Fiabilité à l'issu du nombre important de nœuds
- Calcul de long durée :

  $MTBF <$ temps d'exécution de l'application

Probabilité de défaillance dans les plates-formes de grande taille

Calcul sur grappe : processeurs identiques

- Pannes indépendantes
- MTBF (temps moyen entre 2 pannes sur n'importe quel processeur) = 2000 jours



Objectifs

Mécanismes de tolérance aux pannes pour des applications parallèles

- Plates-formes hétérogènes
- Plates-formes dynamiques
- Transparence pour l'utilisateur
- Faible coût à l'exécution

Plan

État de l'art

Modèle de programmation & vol de travail

Modèle d'exécution par graphe de flot de données

Protocole TIC (*Theft-Induced Checkpointing*)

Protocole SEL (*Systematic Event Logging*)

Résultats expérimentaux

Conclusion & perspectives

Plan

État de l'art

Modèle de programmation & vol de travail

Modèle d'exécution par graphe de flot de données

Protocole TIC (*Theft-Induced Checkpointing*)

Protocole SEL (*Systematic Event Logging*)

Résultats expérimentaux

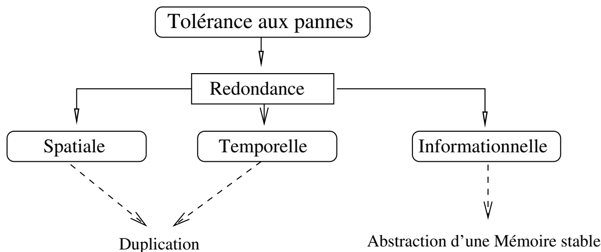
Conclusion & perspectives

Tolérance aux pannes : principes généraux

Modèles de pannes

Franches, par omission, de temporisation, par valeur et byzantines

Technique : redondance



Positionnement

Tolérance aux pannes pour les applications parallèles

- Par duplication
  - Tolère seulement un nombre fixe de pannes
  - Consommation de ressources pour dupliquer le calcul

  ⇒ Non adaptée pour les applications parallèles

Solution retenue : par mémoire stable

Pannes considérées

- pannes franches
- pannes pouvant être ramenées aux pannes franches
  - volatilité de nœuds

Tolérance aux pannes par mémoire stable

Principe

Remplacer l'état erroné du système par un état correct

Ce qui nécessite :

- Identification de l'état erroné : détection de panne
- Recouvrement de panne
  - Sauvegarder l'état du système
  - Restaurer l'état du système

$\Rightarrow$ **Notion d'état global**

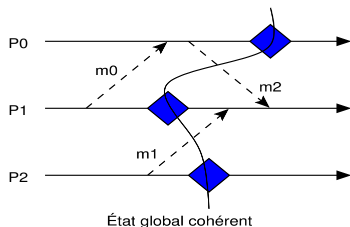
Définition : état global

État global d'un système distribué

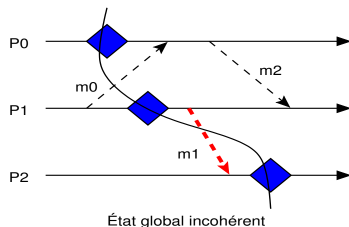
- états locaux de tous les processus du système
- états des canaux de communication entre les processus

État global cohérent

C'est un état global tel qu'il existe une exécution sans panne qui passe par cet état global
(ne contient pas des messages reçus mais non envoyés)



État global cohérent



État global incohérent

Méthodes par mémoire stable : Sauvegarde

Principe

- Sauvegarder des points de reprise pour les processus
- Construction d'un état global cohérent
  - A priori : durant la sauvegarde
  - A posteriori : durant la reprise

Protocoles

- Coordonné : synchronisation globale, pas d'effet domino
- Non coordonné : pas de synchronisation globale, effet domino possible
- Induit par les communications : compromis entre coordonné et non coordonné

Méthodes par mémoire stable : Journalisation

Hypothèse PWD (Piecewise deterministic)

- L'exécution d'un processus est une séquence d'intervalles déterministes d'états
- Les événements non déterministes qui initialisent les intervalles d'états peuvent être identifiés et capturés

Principe

- S'appuie sur l'hypothèse PWD
- La réception d'un message est le seul événement non déterministe
- Sauvegarde de l'histoire de l'exécution
  - Sauvegarde des événements non déterministes

Protocoles : Optimiste, Pessimiste et Causal

Exemples d'environnements parallèles tolérant les pannes

Systèmes représentatifs

- CoCheck : MPI , Condor, sauvegarde coordonnée [Stellner'96]
- MPICH-V2 : MPI, Condor, journalisation des messages [Cappello'03]
- MPICH-CL : MPI, Condor, sauvegarde coordonnée [Bouteiller'03]
- Satin : série parallèle, JAVA, journalisation [Nieuwpoort'05]

Comparaison des systèmes existants

Critères de comparaison	CoCheck	MPICH-V2	MPICH-CL	Satin
information sauvegardée	Image mémoire	Image mémoire	Image mémoire	Tâches orphelines
Multithreading	non	non	non	oui
Hétérogénéité	non	non	non	oui
Remplacement de ressource défaillante	nouvelle ressource	nouvelle ressource	nouvelle ressource	pas besoin
Reprise	globale	locale globle	globale	locale pas globale

Plan

État de l'art

Modèle de programmation & vol de travail

- Modèle d'exécution par graphe de flot de données
- Protocole TIC (*Theft-Induced Checkpointing*)
- Protocole SEL (*Systematic Event Logging*)
- Résultats expérimentaux

Conclusion & perspectives

Modèle de programmation & vol de travail

Programmation parallèle haut niveau

- Description implicite du parallélisme
  - tâches et dépendances de données
  - graphe de flot de données
- Avantages :
  - Applications portables
  - Facilité la programmation parallèle
- Ex. : Cilk, Charm++, Menta, Satin, ATHAPASCAN/KAAPI

Ordonnancement

- Vol de travail
- Prouvé efficace en théorie avec des ressources hétérogènes [Bender'02,...]

Ordonnancement par vol de travail

Principe

- Chaque processeur gère localement la liste des tâches que lui-même a créées ordonnée selon l'ordre séquentiel
- Lorsqu'un processeur termine une tâche :
  - Soit sa liste contient des tâches prêtes ⇒ exécute la plus prioritaire suivant l'ordre séquentiel
  - Soit sa liste est vide ou ne contient pas de tâches prêtes ⇒ devient voleur et cherche à récupérer du travail sur les autres processeurs selon un ordre parallèle
- Vérifie la propriété glouton :
  - A tout instant où il existe une tâche prête mais non encore ordonnancée, tous les processeurs sont actifs

Algorithme distribué

- Processeur victime choisi au hasard

Modèle de coût associé au vol de travail

Notations (cas processeurs identiques)

- T_s : la durée d'exécution de l'algorithme séquentiel
- T_1 : la durée d'exécution de l'algorithme parallèle sur 1 processeur
- T_∞ : la durée d'exécution du chemin critique
- T_p : la durée d'exécution de l'algorithme parallèle sur p processeurs

Théorème [BL97, GRCD98]

- le temps d'exécution T_p d'un programme avec l'algorithme de vol de travail sur p processeurs est majoré par

$$T_p \leq T_1/p + c_\infty T_\infty$$

- le nombre de vols réussis N_{theft} est majoré avec une grande probabilité par

$$N_{theft} \leq O(pT_\infty)$$

Implantation du vol de travail

Nécessite le calcul des tâches prêtes

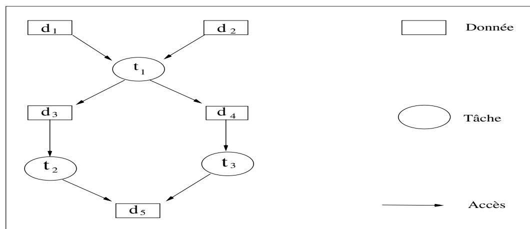
- Représentation abstraite des tâches et de leurs dépendances

⇒ Graphe de flot de données

Le graphe de flot de données associé à l'exécution d'une application : c'est le graphe $G = (V, E)$ tel que $V = V_t \cup V_d$

- V_t : tâches du programme, V_d : données partagées
- E : accès des tâches aux données partagées

∗ Le graphe est dynamique (implicitement distribué)



Contribution

Graphe de flot de données

- État global de l'exécution
- Indépendant du nombre de ressources
- Indépendant de l'architecture

Proposition : utilisation du graphe pour la tolérance aux pannes

Le graphe de flot de données offre la possibilité d'une redondance informationnelle suffisante pour réaliser un mécanisme de tolérance aux pannes dans un environnement hétérogène et dynamique

Plan

État de l'art

Modèle de programmation & vol de travail

Modèle d'exécution par graphe de flot de données

Protocole TIC (*Theft-Induced Checkpointing*)

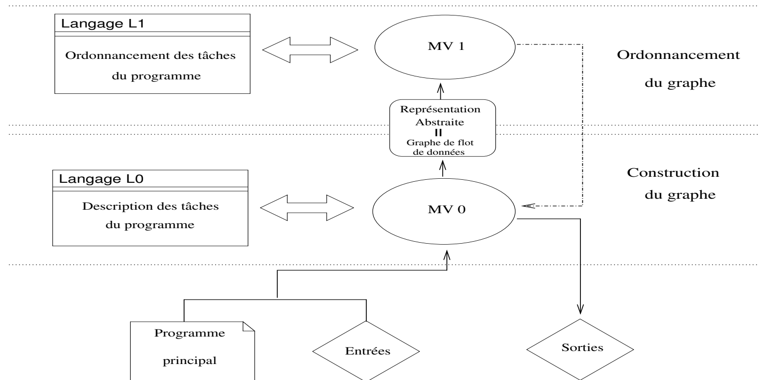
Protocole SEL (*Systematic Event Logging*)

Résultats expérimentaux

Conclusion & perspectives

Modèle d'exécution par graphe de flot de données

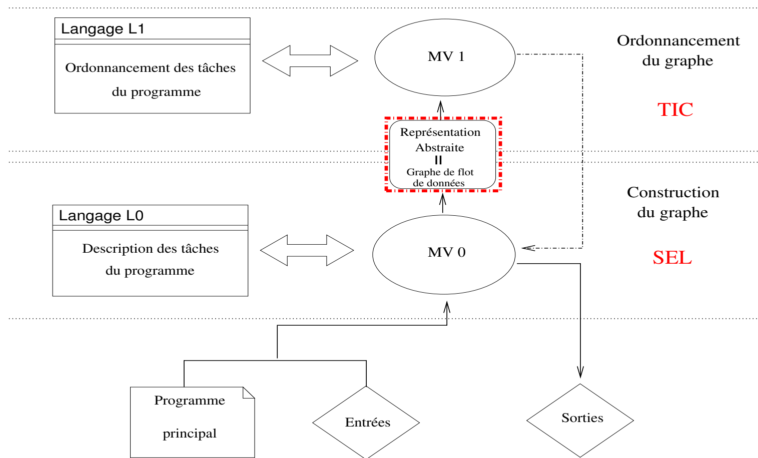
- Graphe dynamique et distribué
- Ordonnancement distribué par vol de travail



Exemple : l'intergiciel KAAPI

Modèle d'exécution & Sauvegarde/Reprise par graphe de flot de données

Proposition : deux protocoles pour capturer le graphe



Plan

État de l'art

Modèle de programmation & vol de travail

Modèle d'exécution par graphe de flot de données

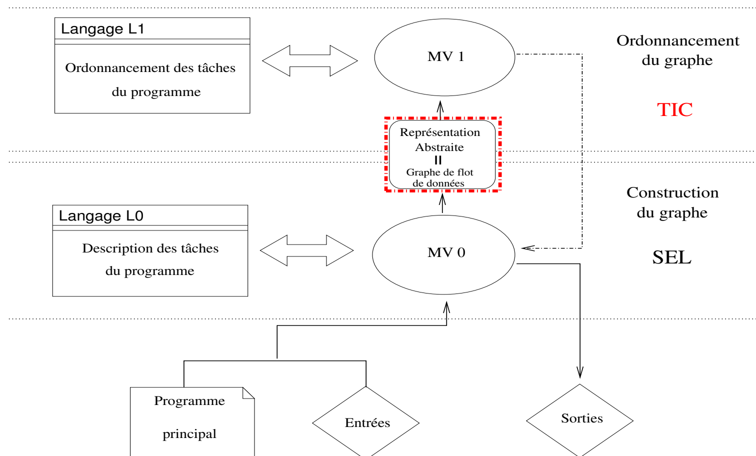
Protocole TIC (*Theft-Induced Checkpointing*)

Protocole SEL (*Systematic Event Logging*)

Résultats expérimentaux

Conclusion & perspectives

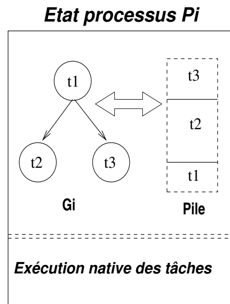
Protocole TIC(*Theft-Induced Checkpointing*)



Protocole TIC

Définition d'un point de reprise

- Un point de reprise concerne un processus
- Le point de reprise d'un processus p_i est sa partie G_i du graphe de flot de données G de l'application
  - Ses tâches et leurs paramètres



Différence importante

- Sauvegarde uniquement des tâches et de leurs paramètres
  $\Rightarrow$ indépendant des plates-formes
- Ne sauvegarde pas l'état d'exécution des tâches
  $\Rightarrow$ le contexte d'un processus dépend de la plate-forme

$\Rightarrow$ **Sauvegarde avant ou après l'exécution d'une tâche**

TIC : Sauvegarde induite par le vol de travail

Idée

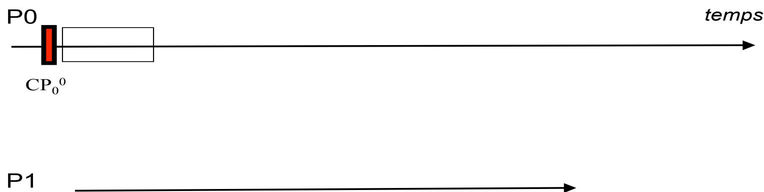
- Méthode de sauvegarde induite par les communications [Bal97]

Principe

- Sauvegarde des points de reprise locaux à l'expiration d'une période τ
- Sauvegarde des points de reprise forcés par une opération de vol de travail

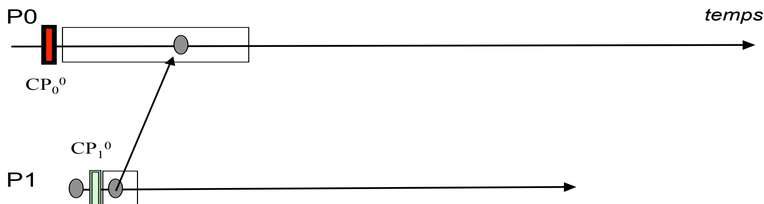
Protocole TIC : Sauvegarde

- CP_i^j le j^e point de reprise du processus P_i



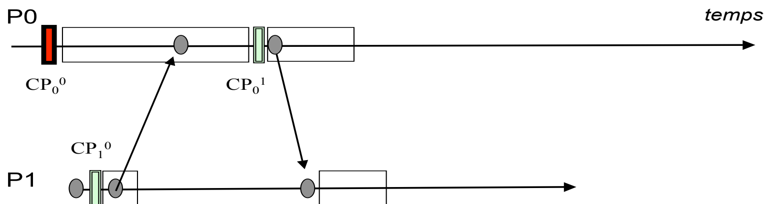
Protocole TIC : Sauvegarde

- CP_i^j le j^e point de reprise du processus P_i



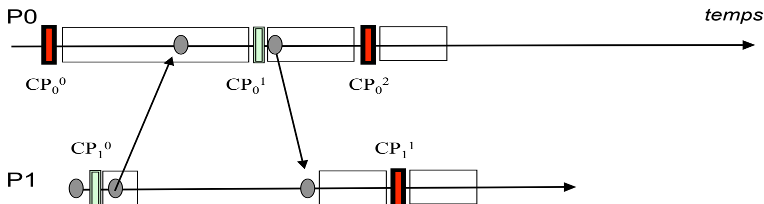
Protocole TIC : Sauvegarde

- CP_i^j le j^e point de reprise du processus P_i



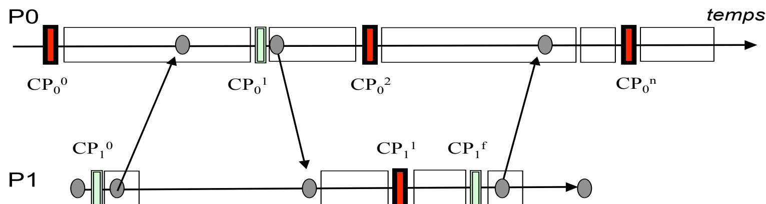
Protocole TIC : Sauvegarde

- CP_i^j le j^e point de reprise du processus P_i



Protocole TIC : Sauvegarde

- CP_i^j le j^e point de reprise du processus P_i



Protocole TIC : Reprise

Permet une reprise locale ou globale

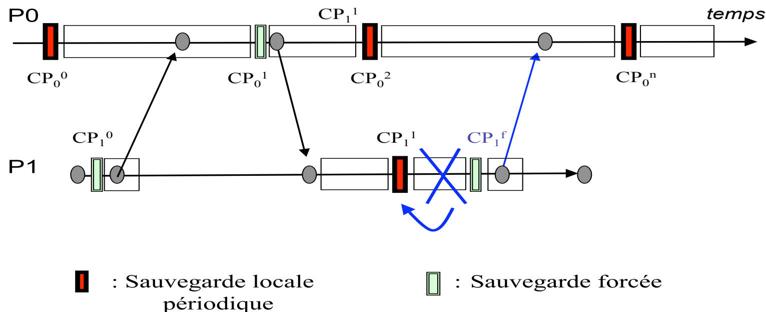
- Reprise = interpréter et rejouer les points de reprise
- Preuve de la correction de la reprise : répond aux deux questions
  - Que fait un processus qui a besoin d'envoyer un message à un processus défaillant ?
  - Comment un processus restaurant un processus défaillant peut-il récupérer les messages qui ont été envoyés au processus défaillant après son dernier point de reprise et avant la détection de sa défaillance ?

⇒ Démonstration de 3 cas :

- Panne du voleur (P_1)
- Panne de la victime (P_0)
- Panne de la victime (P_0) et du voleur (P_1)

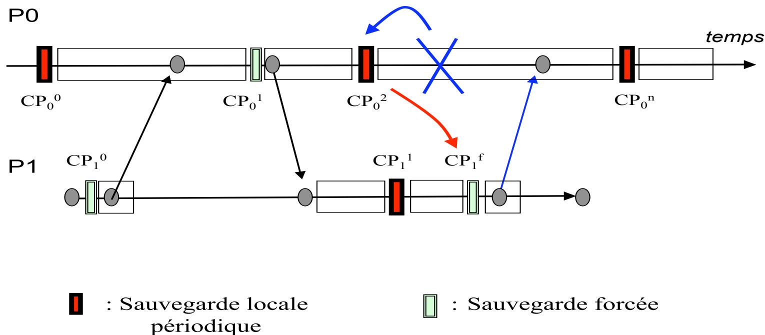
Protocole TIC : Reprise

Cas 1 : panne du voleur (P_1)



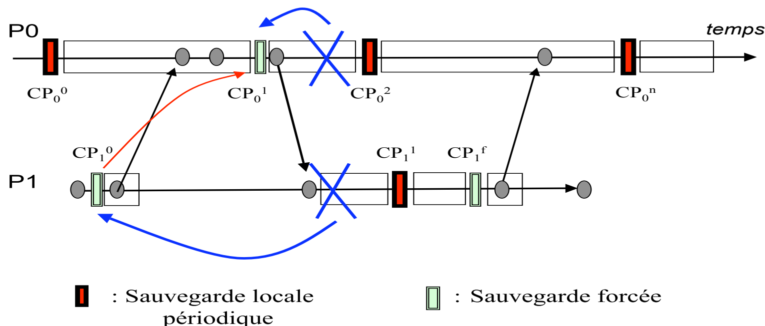
Protocole TIC : Reprise

Cas 2 : panne de la victime (P_0)



Protocole TIC : Reprise

Cas 3 : panne de la victime (P_0) et du voleur (P_1)



Analyse de complexité pour TIC

- Classe des programmes considérés : $T_1 \gg T_\infty$
- t_s : accès élémentaire aux support de stockage
- N_∞ : # maximum de tâches sur un plus long chemin du graphe ($N_\infty = O(T_\infty)$)

TIC dépend du nombre de vols réussis et période

$$T_P^{TIC} \leq T_p + [T_P^{TIC}/\tau + N_{theft}]\ f_{overhead}^{TIC}(N_\infty, t_s)$$

Coût reprise en pire cas

$T_{reprise} \leq O(N_\infty) + \tau +$ durée maximum d'exécution 1 tâche

Calcul perdu après une panne (pire cas)

Période de sauvegarde + le temps d'exécution d'une tâche

TIC : Conclusion

Apports du protocole TIC

- Seuls les processus défaillants font une reprise à partir de leurs derniers points de reprise
- Pas besoin d'une nouvelle ressource pour la reprise d'une ressource défaillante

Comment réduire le temps de calcul perdu suite à une panne ?

- Diminution de la période de sauvegarde
- Limitation du temps de calcul perdu à la durée de l'exécution de la tâche en cours

$\Rightarrow$ Protocole SEL (*Systematic Event Logging*)

Plan

État de l'art

Modèle de programmation & vol de travail

Modèle d'exécution par graphe de flot de données

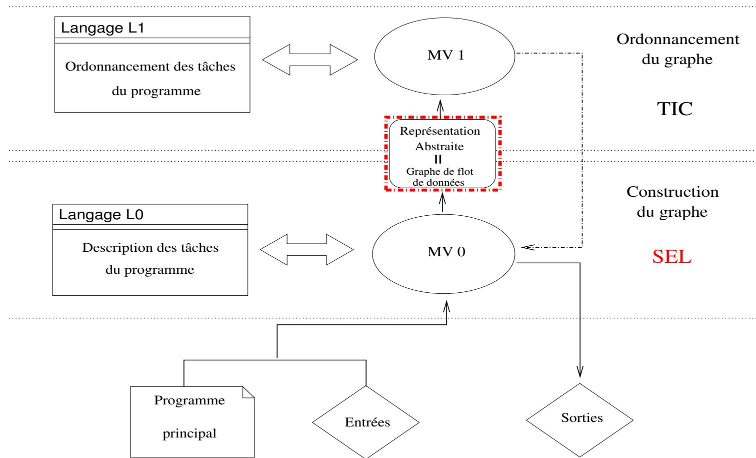
Protocole TIC (*Theft-Induced Checkpointing*)

Protocole SEL (*Systematic Event Logging*)

Résultats expérimentaux

Conclusion & perspectives

Protocole SEL (*Systematic Event Logging*)



Journalisation du graphe de flot de données

Principe de journalisation

Journalisation sur support stable de tous les événements de construction du graphe de flot de donneés :

- Créations / suppressions des tâches
- Créations / modifications / suppressions des données partagées

$\Rightarrow$ Nécessite l'hypothèse PWD

Propriété

Si les tâches d'une application parallèle vérifient :

H_1 Une tâche s'exécute jusqu'à la fin de son exécution sans synchronisation

H_2 L'exécution des tâches est déterministe

$\Rightarrow$ les processus exécutant cette application respectent l'hypothèse PWD

SEL : Reprise

Principe de reprise

Interprétation des événements du journal pour reconstruire le graphe de flot de données de toutes les tâches qui ne sont pas terminées

Problème

- L'exécution d'une tâche peut créer d'autres tâches

$\Rightarrow$ suite à une reprise une tâche peut être créée plusieurs fois

Solution

- Identification unique et reproductible de tous les nœuds du graphe
- Avant la création d'un nœud, on vérifie s'il existe déjà dans le journal

Analyse de complexité pour SEL

- Classe des programmes considérés : $T_1 \gg T_\infty$
- t_s accès élémentaire aux support de stockage

Coût de la journalisation dépend de la taille du graphe G

$$T_P^{SEL} \leq \frac{T_1^{SEL}}{p} + c_\infty T_\infty$$

$$T_1^{SEL} = T_1 + f_{overhead}^{SEL}(|G|, t_s)$$

Coût reprise en pire cas

$T_{reprise} = O(|G_i|)$ + durée maximum d'exécution 1 tâche
où G_i le graphe du processus défaillant

Calcul perdu après une panne (pire cas)

Au plus le calcul d'une tâche

SEL : Conclusion

Propriété

- Assure qu'une exécution complète de l'application est atteinte après un nombre fini de ré-exécutions si $MTBF >$ temps d'exécution d'une tâche (avec une grande probabilité)

Apports

- Pas besoin d'une nouvelle ressource pour la reprise d'une ressource défaillante
- Temps de calcul perdu est au plus la durée d'exécution d'une tâche

Bilan & comparaison : TIC et SEL

TIC : surcoût lié au nombre de vols et période

- Diminution : augmentation de la période

  $\Rightarrow$ Augmentation de temps perdu après panne

SEL : surcoût lié à la taille du graphe

- Diminution : augmentation du grain du programme

  $\Rightarrow$ Perte de parallélisme

Résultats confirmés expérimentalement par la suite

Plan

État de l'art

Modèle de programmation & vol de travail

Modèle d'exécution par graphe de flot de données

Protocole TIC (*Theft-Induced Checkpointing*)

Protocole SEL (*Systematic Event Logging*)

Résultats expérimentaux

Conclusion & perspectives

Réalisation

Dans l'intergiciel KAAPI

- Projet MOAIS
- 37 000 lignes de code (C++)

Intégration de la tolérance aux pannes dans KAAPI

- TIC et SEL
- N serveurs de points de reprise
- Détecteur de défaillances (simple)
- 10 000 lignes de code (C++)

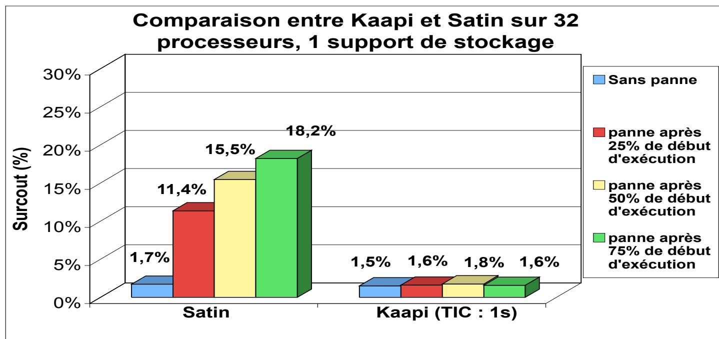
KAAPI tolérant les pannes $\sim$ 47 000 lignes de codes

https ://gforge.inria.fr/projects/kaapi/

Comparaison KAAPI/Satin

Surcoût relatif à l'exécution parallèle

- Application : UB_{Tree}
- Plate-forme (gdx) : 216 Opterons, Bi-processeurs, 2 GHz, mémoire 2 GB, Gigabit ethernet



Validation expérimentale

Application

- ACI DOCG & ARC IMAG-INRIA AHA & CHOC
- Optimisation combinatoire, algorithme Branch & Bound
- QAP : Quadratic Assignment Problem

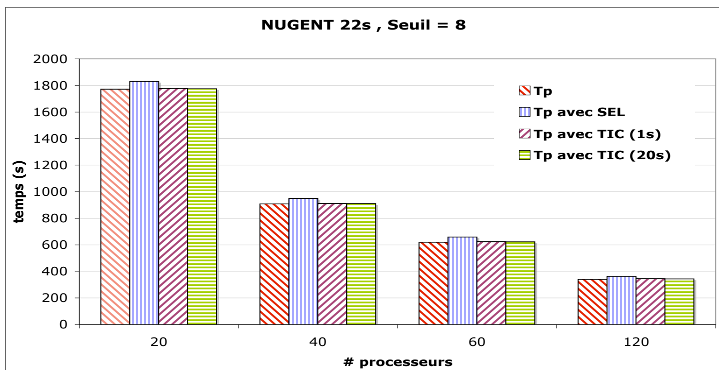
Plate-forme

- iCluster2
  - 104 noeuds Itanium2, Bi-processeurs, 900 MHz
  - Réseau FastEthernet 100 Mbit/s
- Grid5000

Influence du nombre de processeurs

Coût de sauvegarde avec SEL et TIC

$\#$ supports de stockage $=$ $\#$ processeurs



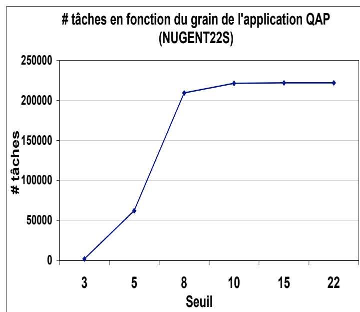
$\Rightarrow$ Surcoût faible

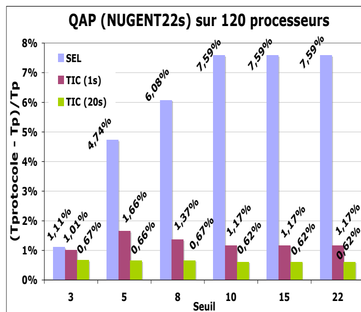
$\Rightarrow$ Passage à l'échelle : support de stockage distribué

Influence du grain de l'application

Surcoût de sauvegarde (SEL et TIC)

QAP (NUG22S) sur 120 processeur



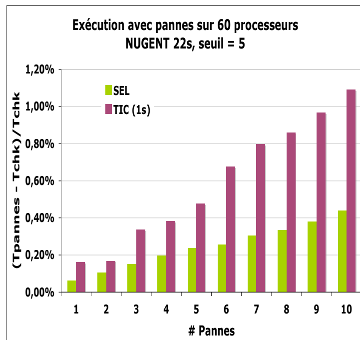


A grain fin le surcoût de SEL augmente

A gros grain le surcoût de TIC augmente

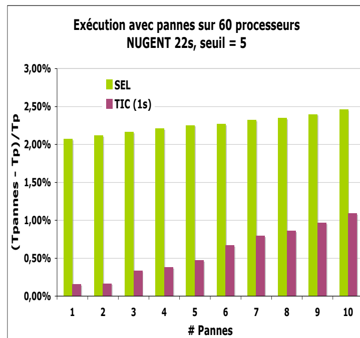
Influence du nombre de pannes

Surcoût reprise



Reprise : $SEL < TIC$

Surcoût total

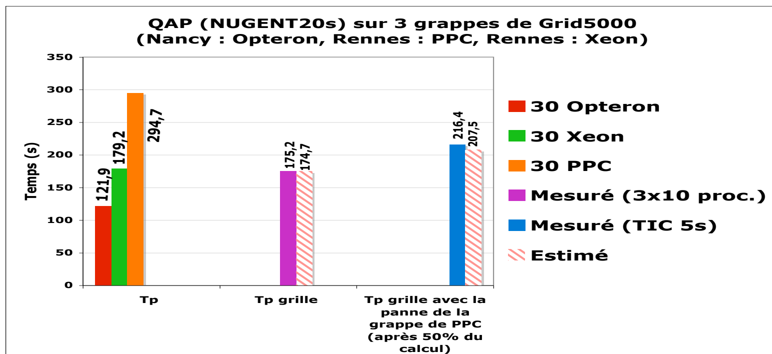


Surcoût total $< 2.5\%$

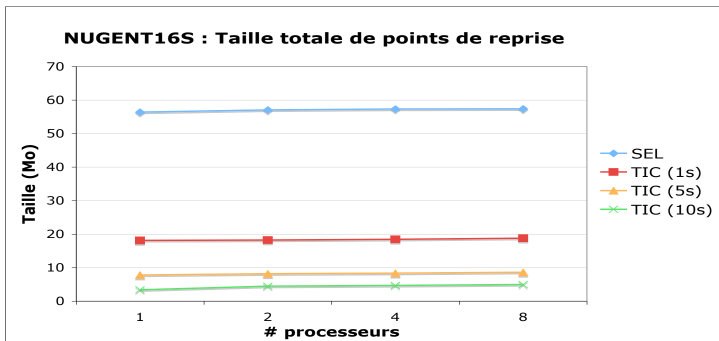
Dynamicité & hétérogénéité

Expérience sur 3 grappes hétérogènes de Grid5000

- Lancement de l'exécution sur 3 grappes : Grillon (Opteron), Paraci (Xeon) et Tartopom (PPC)
- Support de stockage : la grappe Idpot (Xeon, Grenoble), 3 supports
- Retrait de la grappe Tartopom après 50% de début de l'exécution



Volume total des données enregistrées sur support stable



⇒ Volume total des données enregistrées est quasi-indépendent du nombre de processeurs (dependence $= O(pT_\infty)$ avec T_∞ très petit)

Intérêt

Dimensionnement possible du support stable

Autres applications des protocoles TIC et SEL [ANR Safescale]

Adaptabilité [ICTTA'06]

- Adaptation à l'ajout de ressources : vol de travail
- Adaptation au départ de ressources : sauvegarde / reprise

⇒ Utilisation des réservations "BestEffort"

Certification des applications parallèles [DEXA'04, EGC'05, EIT'05]

- Pannes par valeur : attaques malicieuses
- Ré-exécution de tâches sur machines sûres

⇒ SEL pour détecter les pannes par valeur

Plan

État de l'art

Modèle de programmation & vol de travail

Modèle d'exécution par graphe de flot de données

Protocole TIC (*Theft-Induced Checkpointing*)

Protocole SEL (*Systematic Event Logging*)

Résultats expérimentaux

Conclusion & perspectives

Conclusion

Deux protocoles de tolérance aux pannes

- Applications parallèles : $T_1 \gg T_\infty$
- Supportent l'hétérogénéité, le multithreading, la dynamicité
- Analyse des surcoûts théoriques
- Réalisation TIC et SEL dans KAAPI
- Validations expérimentales avec l'application QAP
- Application à l'adaptabilité et la certification

Publications

- 5 conférences internationales, 5 conférences nationales

Collaborations

- A. Djerrah, T. Gautier, A. Krings, JL. Roch, S. Varrette

Perspectives

Autre classe d'applications

- Applications itératives
- Communication fréquente

⇒ Extension d'un protocole coordonnée (CCK)

  - Diminue le coût de la coordination globale
  - Seuls les processus défaillants font la reprise

Implantation mémoire stable sur machines non fiables

Protocole adaptatif

- Protocole hybride
- Changer le protocole de tolérance en fonction :
  - Comportement de l'algorithme à l'exécution
  - Comportement de la plate-forme de l'exécution
  - Propriétés attendues

Relaxation du critère de cohérence globale

Merci pour votre attention

Questions ?

R. Baldoni.
A communication-induced checkpointing protocol that ensures rollback-dependency trackability.
In *Proceedings of the 27th International Symposium on Fault-Tolerant Computing (FTCS '97)*, page 68. IEEE Computer Society, 1997.

R.D. Blumofe and C.E. Leiserson.
Space-efficient scheduling of multithreaded computations.
SIAM Journal on Computing, 1(27) :202–229, 1997.

F. Galilée, J.-L. Roch, G. Cavalheiro, and M. Doreille.
Athapascan-1 : On-line building data flow graph in a parallel language.
In IEEE, editor, *PACT'98*, pages 88–95, Paris, France, October 1998.