Adaptation, Sûreté de fonctionnement et certification des résultats des programmes parallèles

Vincent Danjean, Thierry Gautier, **Jean-Louis Roch**, Daouda Traore,
Sébastien Varrette et aussi Samir Jafar

MOAIS project http://moais.imag.fr

¥Projet MOAIS (CNRS,INPG,INRIA,UJF), Laboratoire ID-IMAG, France

+ VASCO (Marie-Laure Potet, Roland Groz)

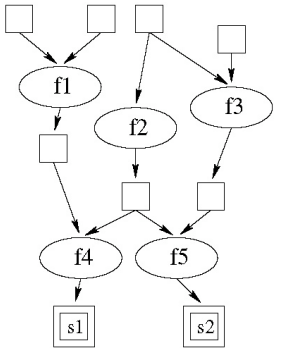
Objectifs MOAIS liés à Safescale

- Programmation **fiable** et **efficace** des architectures de grande taille
  - Grappe, Grille, P2P (et aussi plus petits : SMP, embarqués/MPSoC)
  - Kaapi software

- Nos problèmes cibles :
  - **Adaptation de l'application :**
    #ressources, dynamique (ajout/retrait), hétérogènes, vitesses variables (ressources partagées)
    - => Algorithmes adaptatifs [tâches malléables, poly-algorithmes, …]

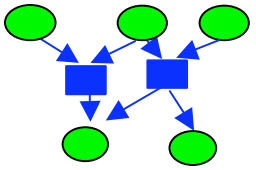
  - **Sécurité / Tolérance aux défaillances**
    - *Accès : authentification distribué, chiffrement coms, …[GRID 5000]*
    - Pannes franches
      - => protocoles sauvegarde/reprise automatique et performants (surcouût faible)
    - Intrusions malicieuses (attaques massives)
      - => Détection d'attaques massives et certification probabiliste
      - => Algorithmes tolérants un nombre donné de défaillances

- Notre approche : ordonnancement + algorithme
  un état de l'application ⇔ un graphe de flot de données
  - Dynamique, distribué
  - « Construit » et ordonnancé à la volée



KAAPI - Modèle d'exécution

- **KAAPI K**ernel for **A**daptive, **A**synchronous **P**arallel **I**nterface
  - http://moais.imag.fr/kaapi (forge inria)
  - Exécution: basée sur une pile locale sur chaque processeur
    - » Macrodataflow dynamique (récursif)
  - Ordonnancement par vol de travail sur inactivité (workstealing)



- Librairie C++ (API/runtime)
  - Support d'APIs de + haut niveau : Athapascan, Homa/Corba, …
- Fondement algorithmique :

  Mesures de coût :sr le graphe à grain « fin »

  T_1 = temps sur 1 proc. ; T_∞ = temps minimal sur ∞ proc. ;

  - T_p = temps sur p proc incluant coût d'ordonnancement
  - Théorèmes fondamentaux :
    - » Si processeurs homogènes $T_p \leq T_1/p + O(T_\infty)$ [Cilk,Athapascan]
    - » Si processeurs hétérogènes : $T_p \leq c_1 T_s/p^* + O(T_\infty)$ [Bender&Rabin]
  - Classe des programmes considérés : $T_\infty << T_1$
    - » Beaucoup de tâches de grain fin possible => **peu de tâches créées**, et elle sont de gros grain ! (O(p T_∞)
    - » pratique : contexte calcul sur architecture de calcul global

Presentation Outline

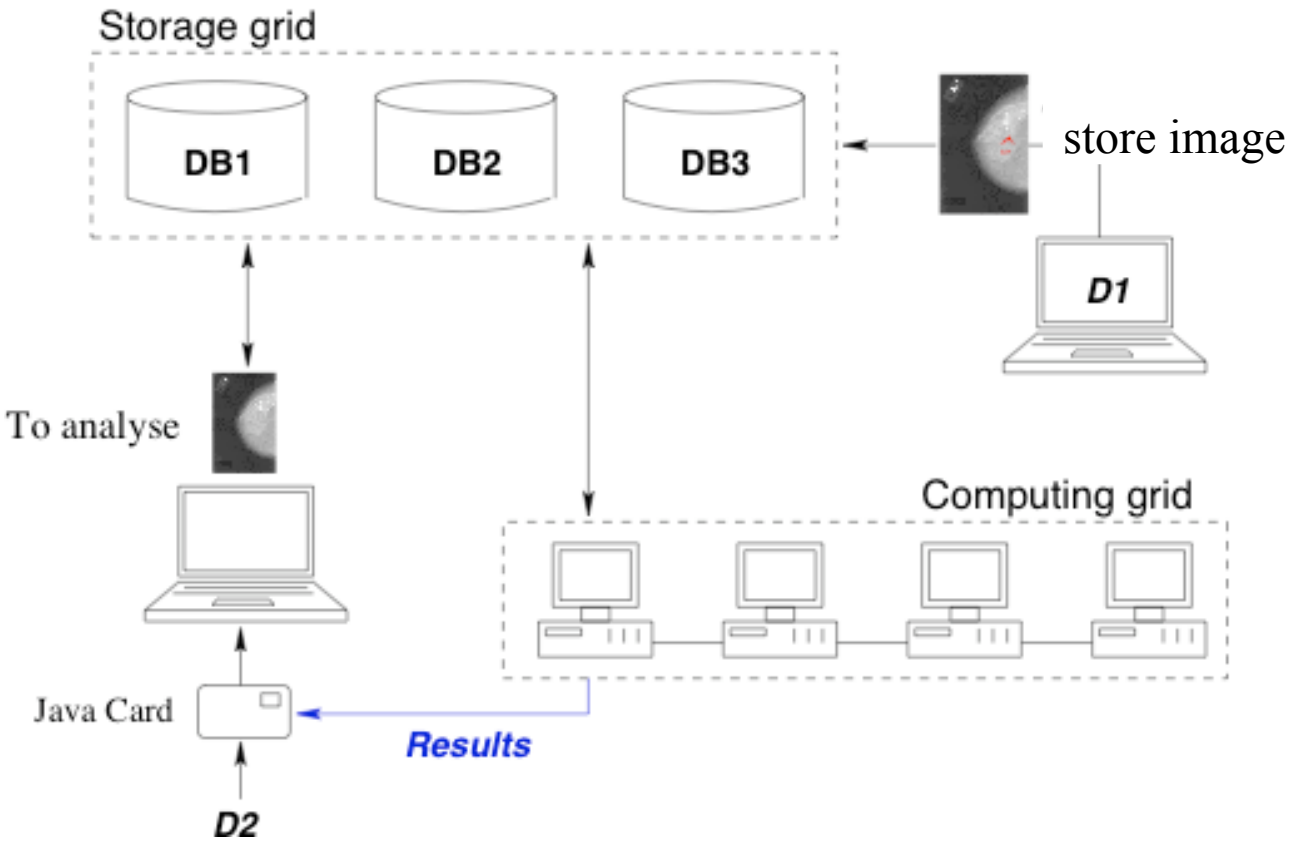
- **Application and Threat**
- Fail stop faults : checkoint/restart
- Massive attacks: probabilistic certification
- Some results

Target Application

- Large-Scale Global Computing Systems
- Subject Application to Dependability Problems
  - Can be addressed in the design
- Subject Application to Security Problems
  - Requires solutions from the area of survivability, security, fault-tolerance

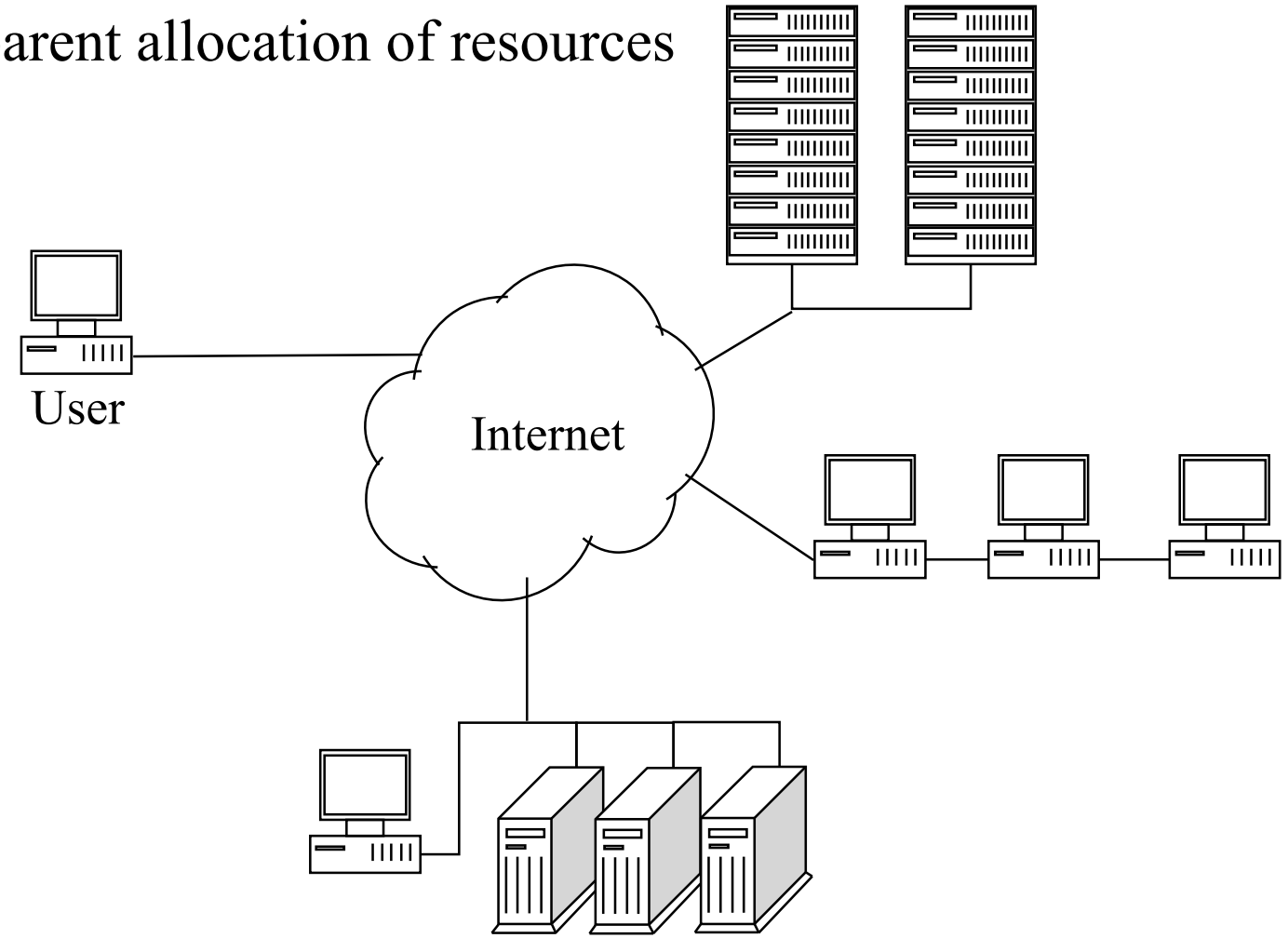
Typical Application [RAGTIME]

- Computation intensive parallel application
  - Medical (mammography comparison)



Global Computing Architecture

- Large-scale distributed systems (e.g. Grid, P2P)
- Transparent allocation of resources



Unbounded Environments

- In the Survivability Community our general computing environment is referred to as

  Unbounded Environment

  - Lack of physical / logical bound
  - Lack of global administrative view of the system.

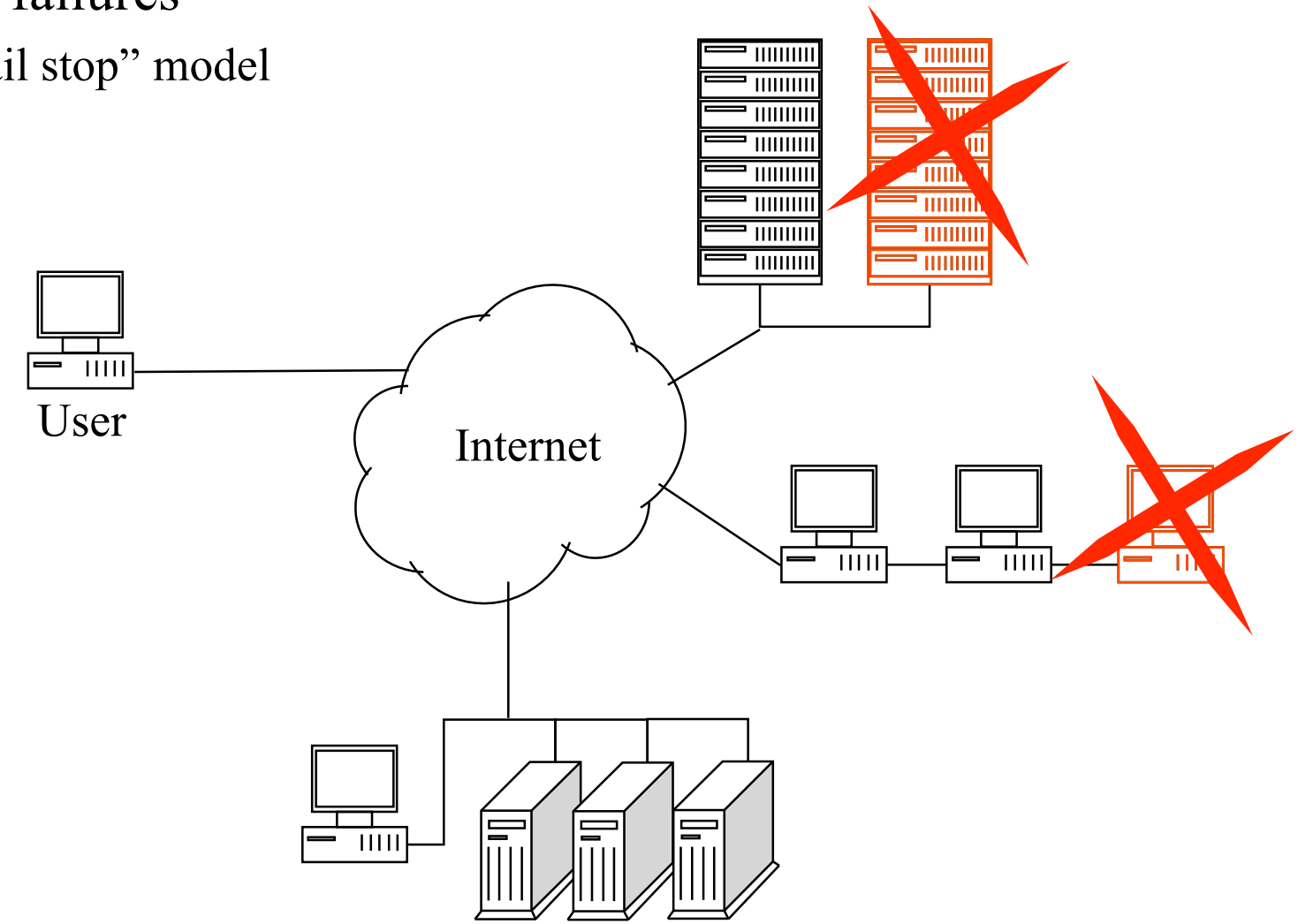
What risks are we subjecting our applications to?

Assumptions

- Anything is possible!
  - » and it <u>will</u> happen!
- Malicious act will occur sooner or later
- It is hard or impossible to predict the behavior of an attack

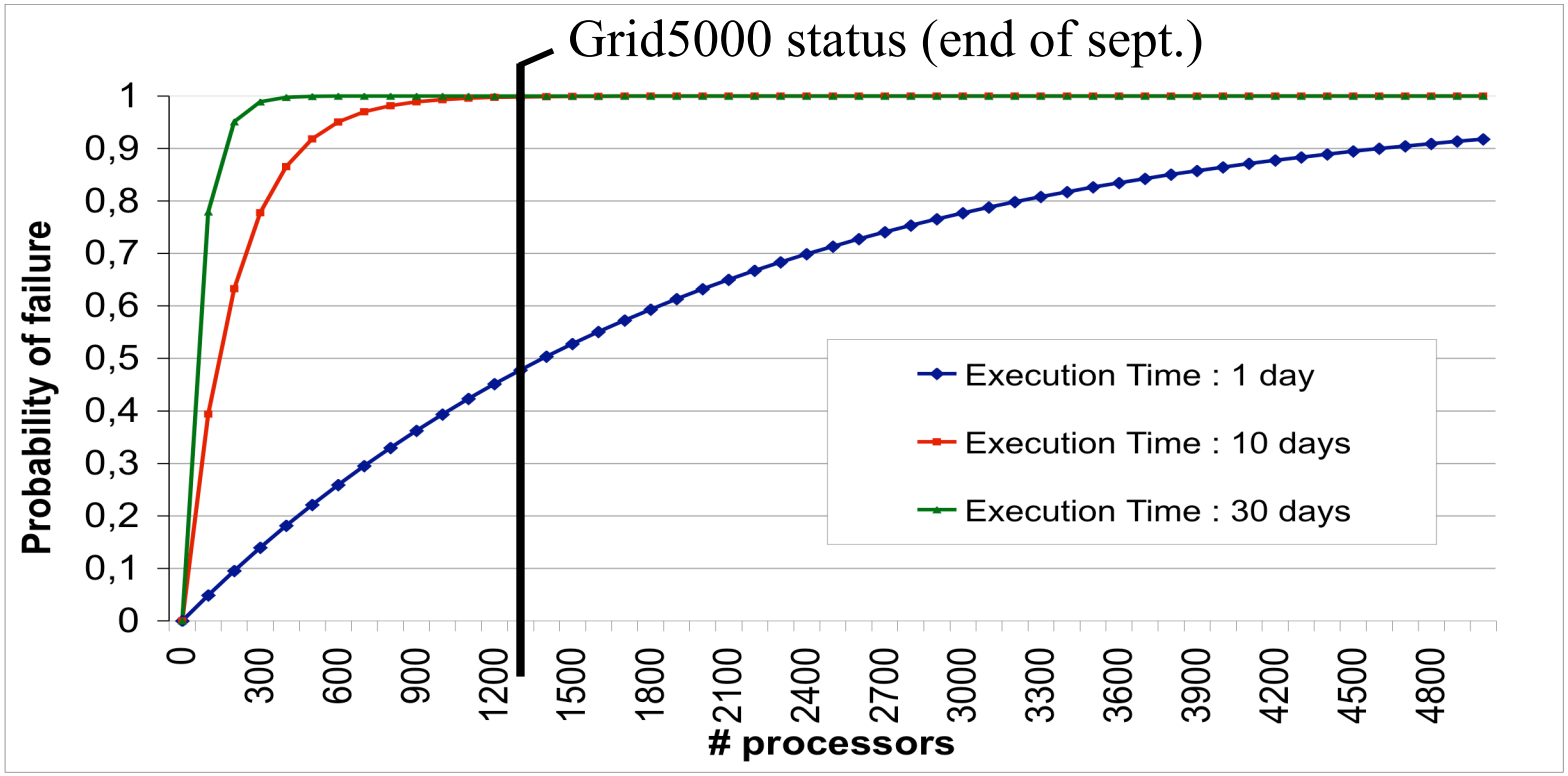
Two kinds of failures (1/2)

1. Node failures
   - "fail stop" model



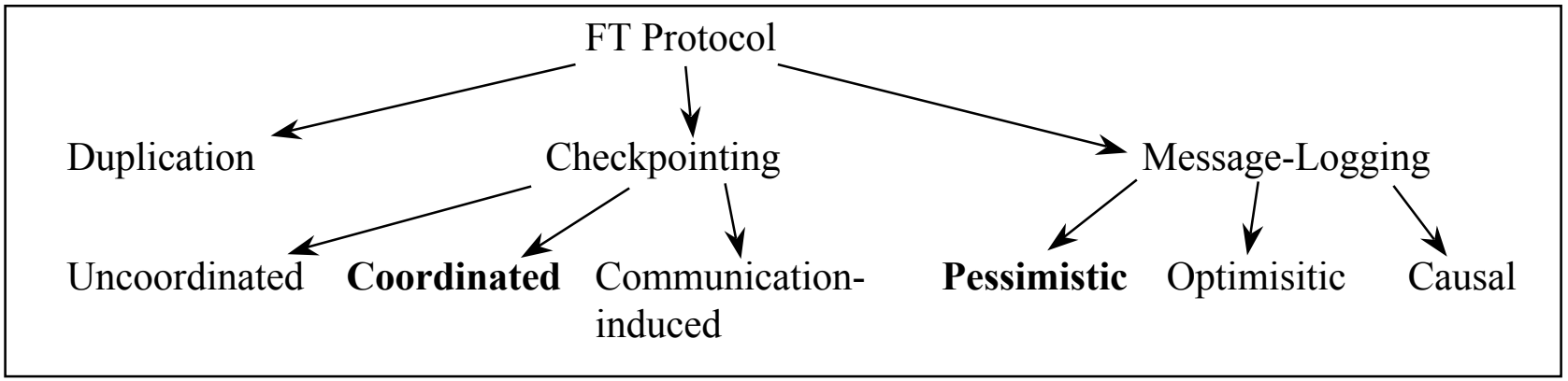
Unreliability in the absence of Fault Tolerance Mechanism

- Computation on Cluster
  - MTBF = 2000 days (48,000h, approx. 5 1/2 years)
  - Unreliability of one node: $F(t) = 1 - R(t) = 1 - e^{-\lambda t}$



Fault Tolerance Approaches

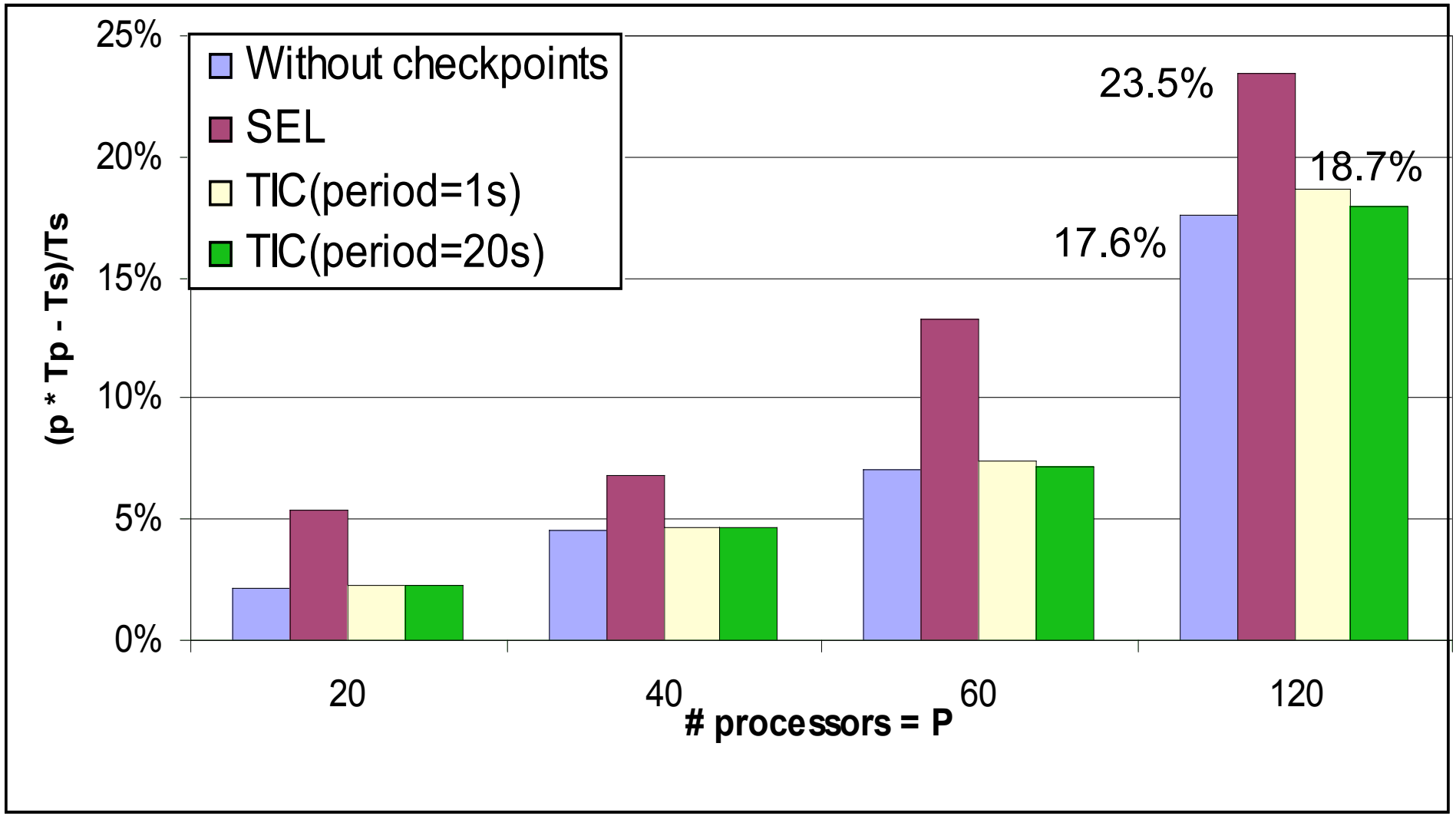
- Simplified Taxonomy for Fault Tolerance Protocols



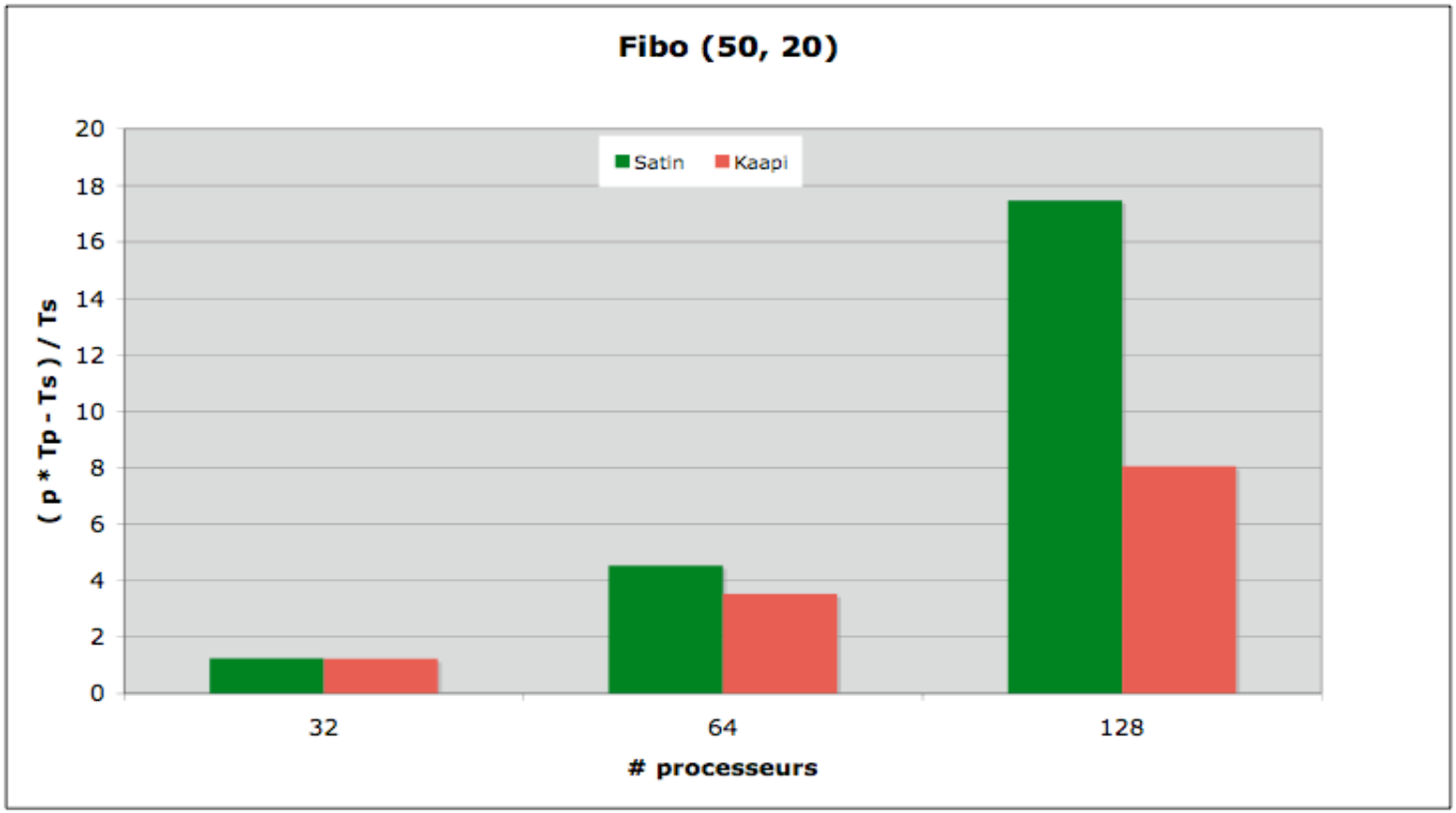
- Rely on a "stable storage"
  - persistent and assumed to be reliable [Kaapi / Athapascan]
  - If not persistent: only duplication of saved data (checkpoint / message)
    - probabilistic FT protocols: fault tolerance is guaranteed with good probability

- Two protocols, distributed, [asynchronous] :
  - SEL : systematic
  - TIC : checkpoint of the local stack when steal + periodic

Passage à l'échelle ?

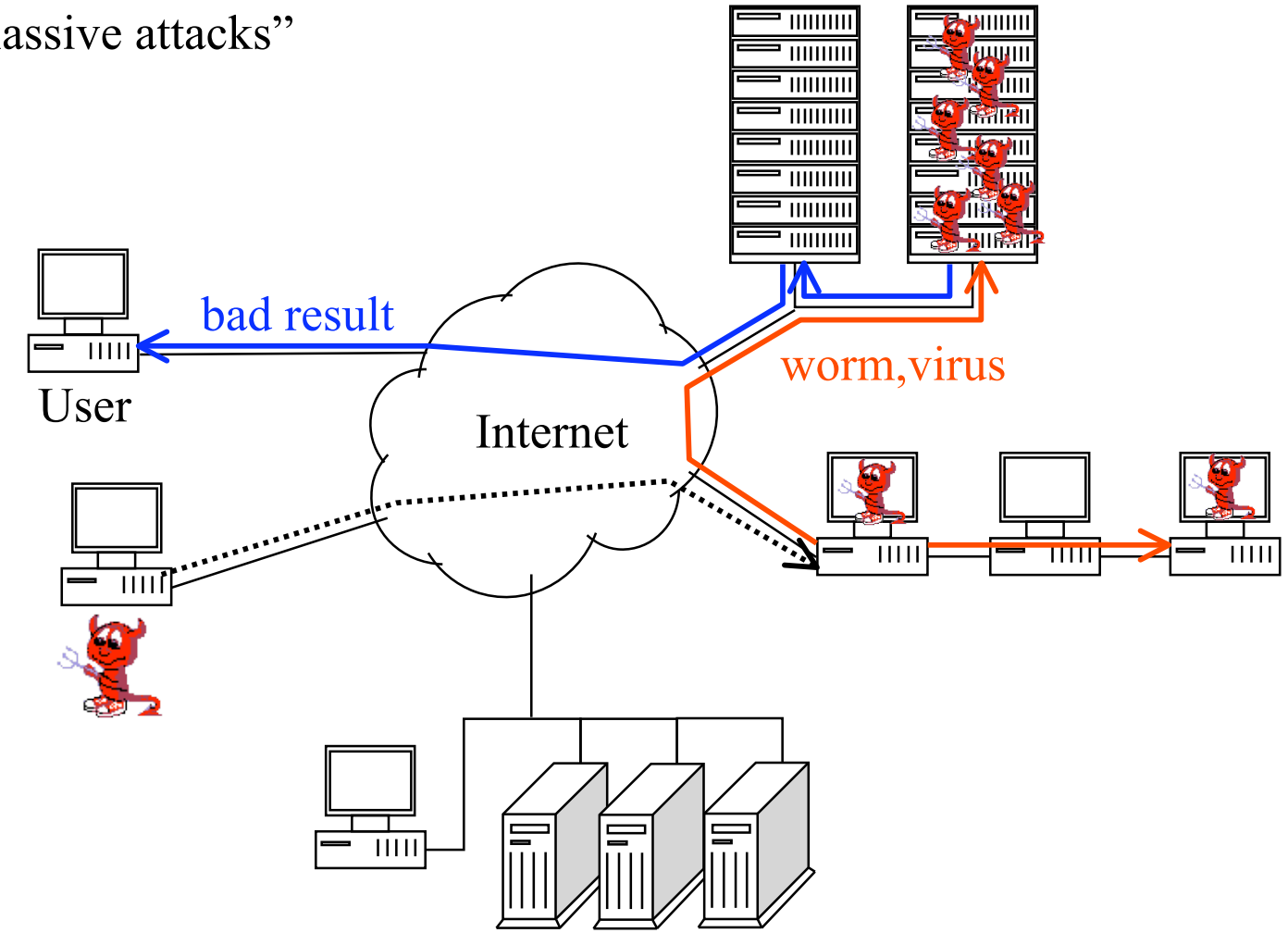


Kaapi / Satin


Fibo (50, 20)
Satin
Kaapi
(p * Tp - Ts) / Ts
0
2
4
6
8
10
12
14
16
18
20
32
64
128
processeurs

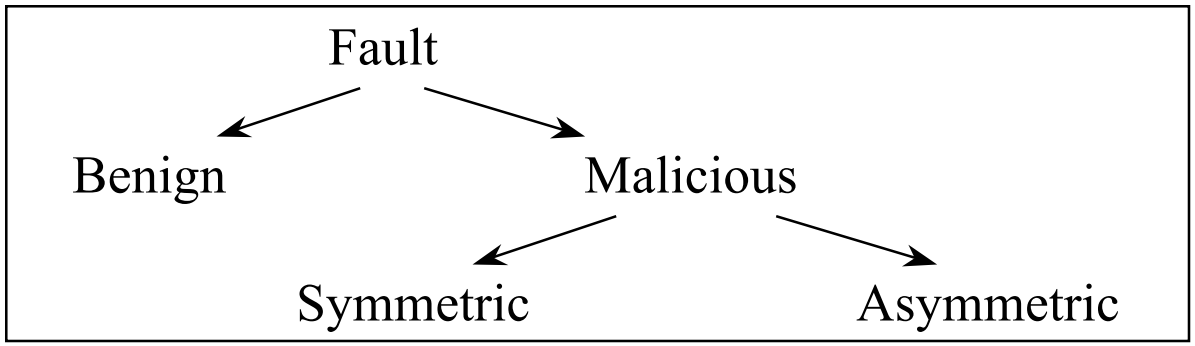
Two kinds of failures (2/2)

2. Task forgery
   - "massive attacks"



Fault Models

- Simplified Fault Taxonomy



- Fault-Behavior and Assumptions
  - Independence of faults
  - Common mode faults -> towards arbitrary faults!

- Fault Sources
  - Trojan, virus, DOS, etc.
  - How do faults affect the overall system?

Attacks and their impact

- Attacks
  - single nodes, difficult to solve with certification strategies
  - solutions: e.g. intrusion detection systems (IDS)
- Massive Attacks
  - affects large number of nodes
  - may spread fast (worm, virus)
  - may be coordinated (Trojan)
- Impact of Attacks
  - attacks are likely to be widespread within neighborhood, e.g. subnet
- Our focus: massive attacks
  - virus, trojan, DoS, etc.

Certification Against Attacks

- Mainly addressed for **independent tasks**

- Current approaches
  - Simple checker [Blum97]
  - Voting [SETI@home]
  - Spot-checking [Germain-Playez 2003, based on Wald test]
  - Blacklisting
  - Credibility-based fault-tolerance [Sarmenta 2003]
  - Partial execution on reliable resources (partitioning) [Gao-Malewicz 2004]
  - Re-execution on reliable resources

- Certification of Computation

Presentation Outline

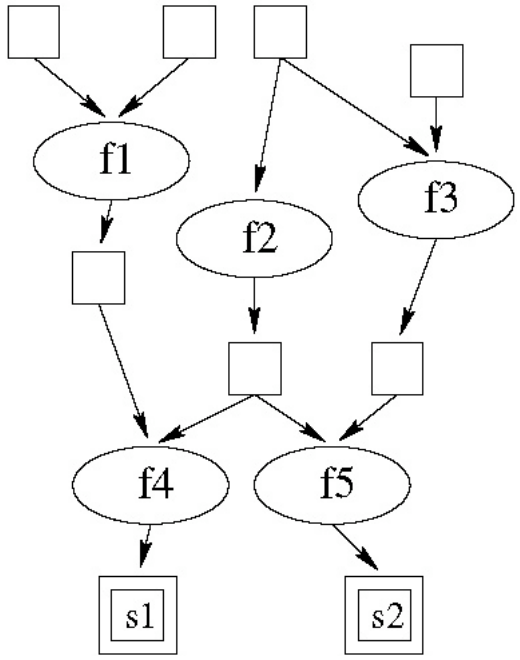
- Motivation: Application and Threat
- **Execution Model**
- Certification with independent tasks
- Certification with task dependencies
- Results
- Conclusions and Future Work

Definitions and Assumptions

- Dataflow Graph
  - $G = (\mathcal{V}, \mathcal{E})$

    $\mathcal{V}$ finite set of vertices v_i

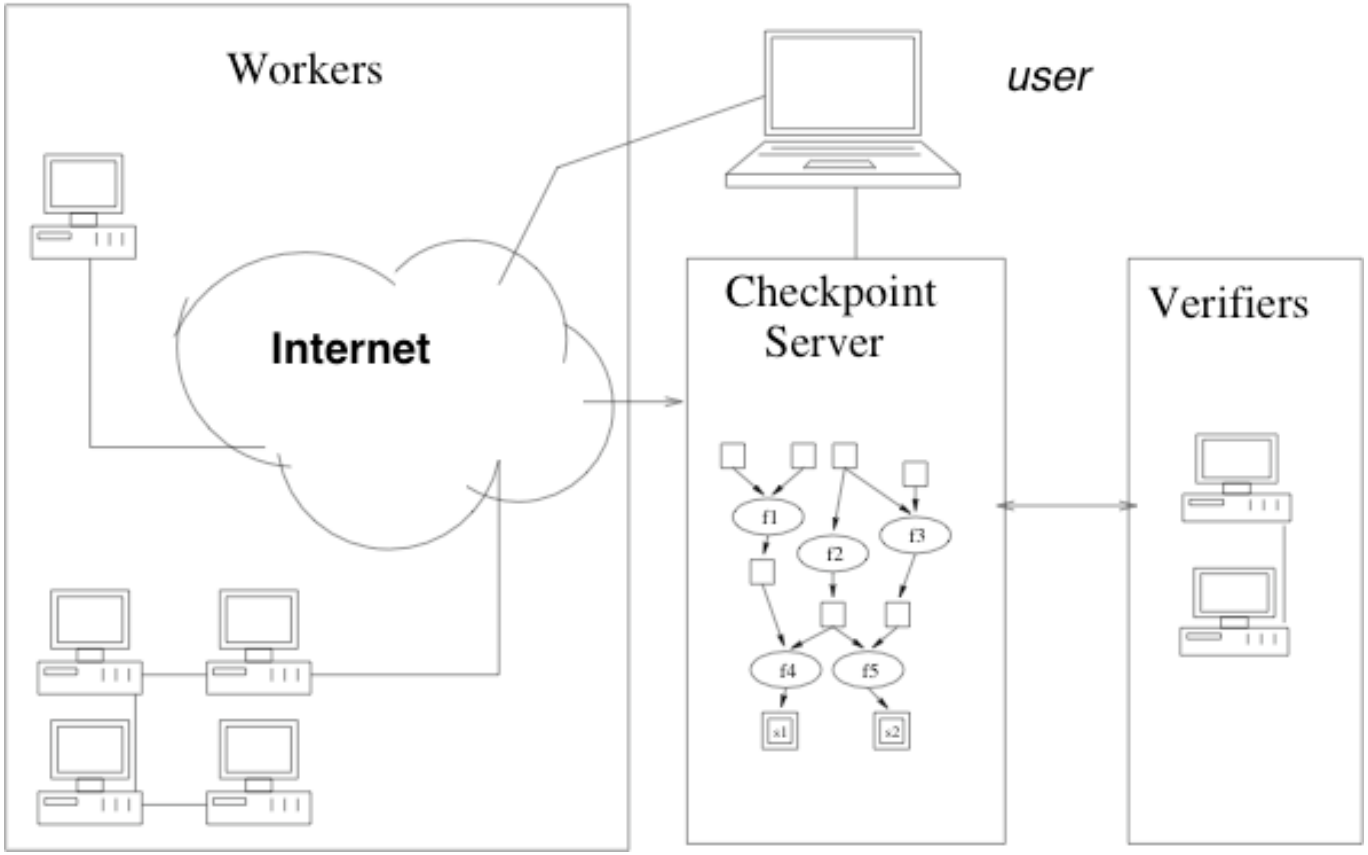
    $\mathcal{E}$ set of edges e_{jk} vertices $v_j , v_k \in \mathcal{V}$



- Two kinds of tasks
  - T_i Tasks
    in the traditional sense
  - D_j Data tasks
    inputs and outputs

Global Computing Platform (GCP)

- GCP includes workers, checkpoint server and verifiers



Probabilistic Certification

- Monte Carlo certification:
  - a randomized algorithm that
    1. takes as input *E* and an arbitrary ε, $0 < \varepsilon \leq 1$
    2. delivers
       - either CORRECT
       - or FAILED, together with a proof that *E* has failed
  - certification is with error ε if the probability of answer CORRECT, when *E* has actually failed, is less than or equal to ε.

- Interest
  - ε : fixed by the user (tunable certification)
  - Number of executions by the verifiers is not to large with respect of the number of tasks

Protocols MCT and EMCTs

- The Basic Protocol: The Monte Carlo Test (MCT) [SBAC04]

  1. Uniformly select one task T in G
     we know input $i(T,E)$ and output $o(T,E)$ of T from checkpoint server
  2. Re-execute T on verifier, using $i(T,E)$ as inputs, to get output $\hat{o}(T,E)$
     If $o(T,E) \neq \hat{o}(T,E)$ return FAILED
  3. Return CORRECT

- Results about extended MCT (EMCTs) [EIT-b 2005]
  - Number N of re-execution depends

$$N \geq \left\lceil \frac{\log \varepsilon}{\log(1 - \Lambda_G)} \right\rceil$$

  - where Λ_G depends on the graph structure, the ratio of tasks forgeries and of the protocol
  - E.g.: For massive attack and independent tasks: $\Lambda_G = q$

Certification of Independent Tasks

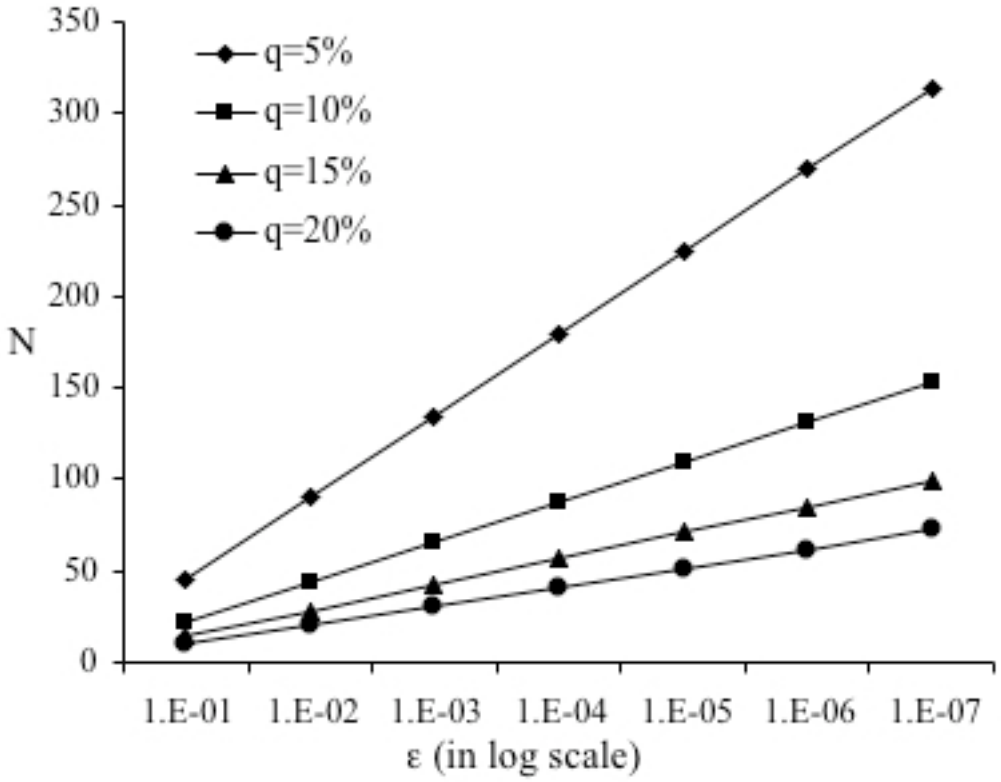
- How many independent executions of MCT are necessary to achieve certification of E with probability of error $\leq \varepsilon$?

$$N \geq \left\lceil \frac{\log \varepsilon}{\log(1-q)} \right\rceil$$

  - Prob. that MCT selects a non-forged tasks is $\frac{n - n_F}{n} \leq 1 - q$
  - N independent applications of MCT results in $\varepsilon \leq (1 - q)^N$

Certification of Independent Tasks

- Relationship between certification error and N



For q = 1% :

•300 checks => $\varepsilon < 5\%$

•4611 checks => $\varepsilon < 10^{-20}$

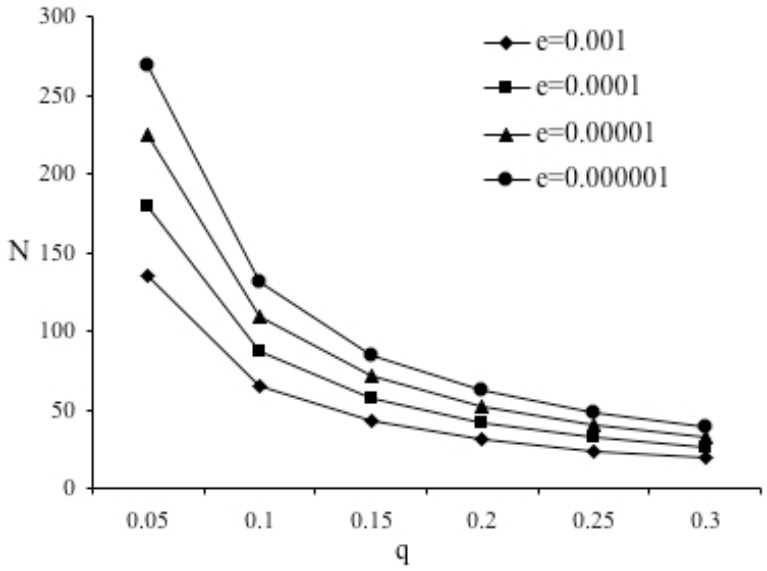
•24000 checks => $\varepsilon < 10^{-125}$

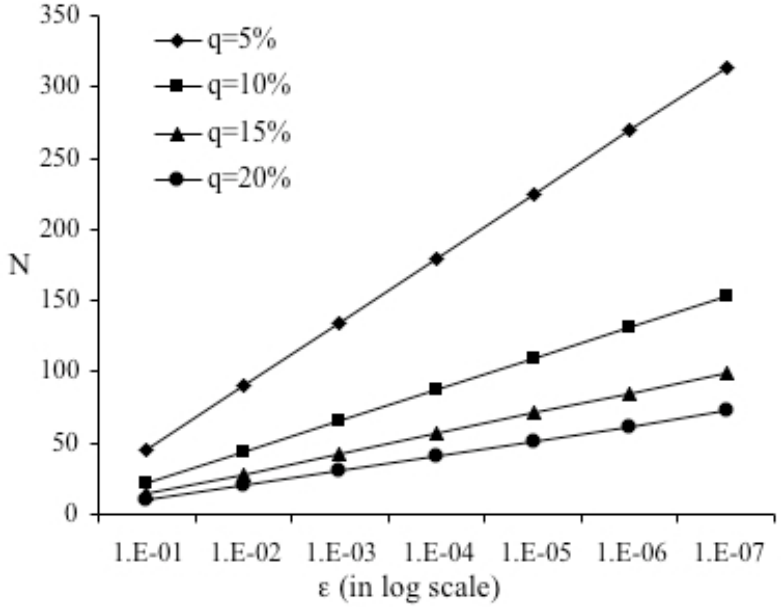
Task dependencies

- Algorithm EMCT
  1. Uniformly select one task T in G
  2. Re-execute all T_j in $G^{\leq}(T)$, which have not been verified yet, with input $i(T,E)$ on a verifier and return FAILED if for any T_j we have $o(T_j,E) \neq \hat{o}(T_j,E)$
  3. Return CORRECT
- Behavior
  - disadvantage: the entire predecessor graph needs to be re-executed
  - however: the cost depends on the graph
    - luckily our application graphs are mainly trees

Analysis of EMCT

- Results of independent tasks still hold,
  - but *N* hides the cost of verification
    - » independent tasks: C = 1
    - » dependent tasks: C = $|G^{\leq}(T)|$





Reducing the cost of verification

For EMCT the entire predecessor graph had to be verified
To reduce verification cost two approaches are considered next:

1. Verification with fractions of $G^{\leq}(T)$
2. Verification with fixed number of tasks in $G^{\leq}(T)$

Results for pathological cases

- Number of effective initiators
  - this is the # of initiators as perceived by the algorithm
  - e.g. for EMCT an initiator in $G^{\leq}(T)$ is <u>always</u> found, if it exists

	$MCT(E)$ [7]	$EMCT(E)$ [7]	$EMCT_{\alpha}(E)$	$EMCT^{1}(E)$
# of effective initiators	$\left\lceil \frac{n_q}{\left(\frac{1-d^h}{1-d}\right)} \right\rceil$	n_q	$n_q \alpha \Gamma_T(n_q)$ or n_q	$n_q \Gamma_T(n_q)$
Probability of error	$1 - \frac{\left\lceil \frac{n_q}{\left(\frac{1-d^h}{1-d}\right)} \right\rceil}{n}$	$1-q$	$1 - q\alpha\Gamma_T(n_q)$ or $1-q$	$1 - q\Gamma_T(n_q)$
A priori convergence	$\frac{\log \epsilon}{\log(1 - \frac{\left\lceil \frac{n_q}{\left(\frac{1-d^h}{1-d}\right)} \right\rceil}{n})}$	$\frac{\log \epsilon}{\log(1-q)}$	$\frac{\log \epsilon}{\log(1-q\alpha\Gamma_G(n_q))}$ or $\frac{\log \epsilon}{\log(1-q)}$	$\frac{\log \epsilon}{\log(1-q\Gamma_G(n_q))}$
q_e a priori	$\frac{\left\lceil \frac{n_q}{\left(\frac{1-d^h}{1-d}\right)} \right\rceil}{n}$	q	$q\alpha\Gamma_G(n_q)$ or q	$q\Gamma_G(n_q)$
q_e run-time	$\frac{\left\lceil \frac{n_q}{\left(\frac{1-d^h}{1-d}\right)} \right\rceil}{n}$	q	$q\alpha\Gamma_T(n_q)$ or q	$q\Gamma_T(n_q)$
Verification cost (exact)	1	$\lvert G^{\leq}(T)\rvert$	$\lceil \alpha \lvert G^{\leq}(T)\rvert \rceil$	1
Max. cost (out-tree)	1	h	αh	1

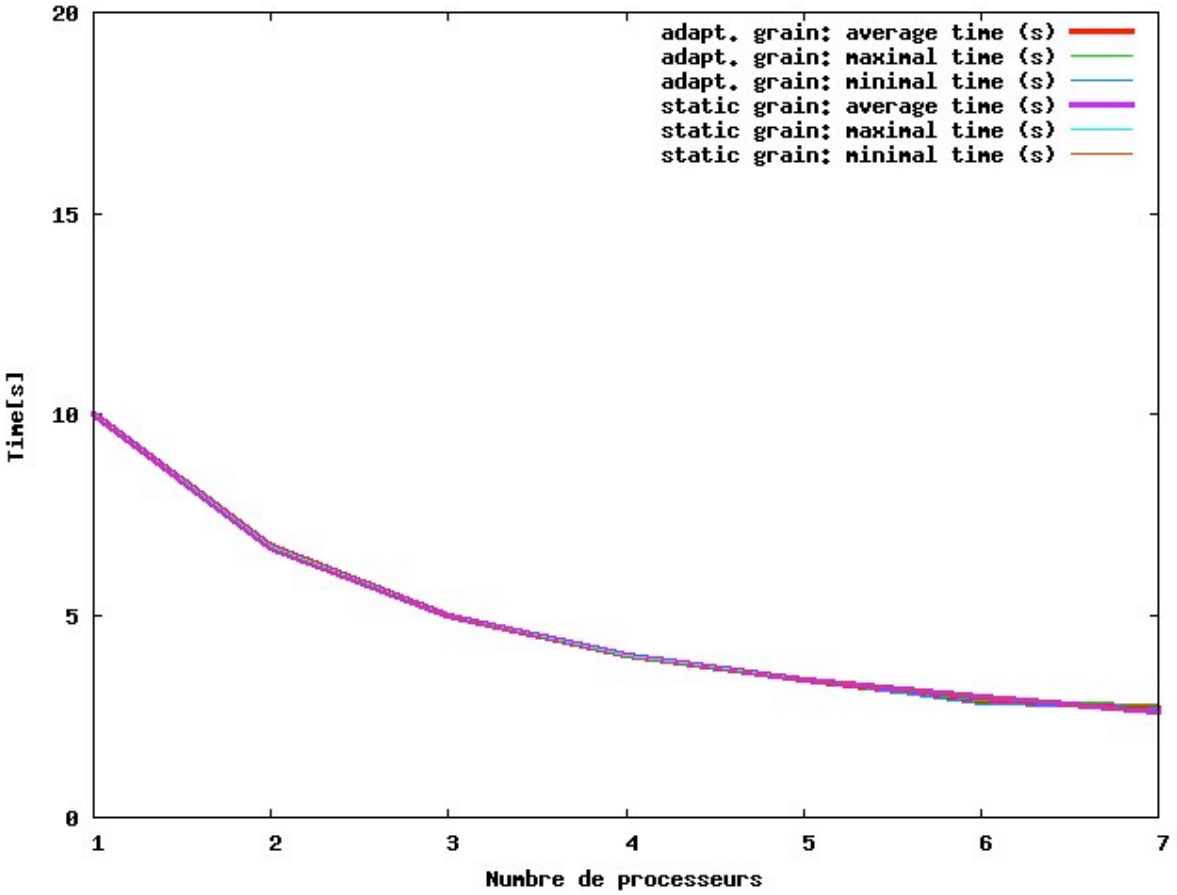
Adaptivité

- Kaapi: réification, interaction avec l'environnement (ajout de ressources), … (interaction)
- Mais aussi : impact sur l'algorithmique / ordonnancement
- Example : workstealing based algorithms
  - Recursive parallel computations
  - Local sequential computation
  - Special case:
    - recursive extraction of parallelism when a resource becomes idle
    - But local execution of a sequential algorithm
- Example : prefix computation
  - Sequential : n operations
  - Parallel on p identical resources : at least 2n.(p/(p+1)) operations
  - Adaptive with work-stealing :
    - Coupling sequential and parallel partial-prefix computation
    - May benefit of an unbounded number or ressources
    - Performance : on p processors of variable speeds :2n/(p+1) + O(log n)

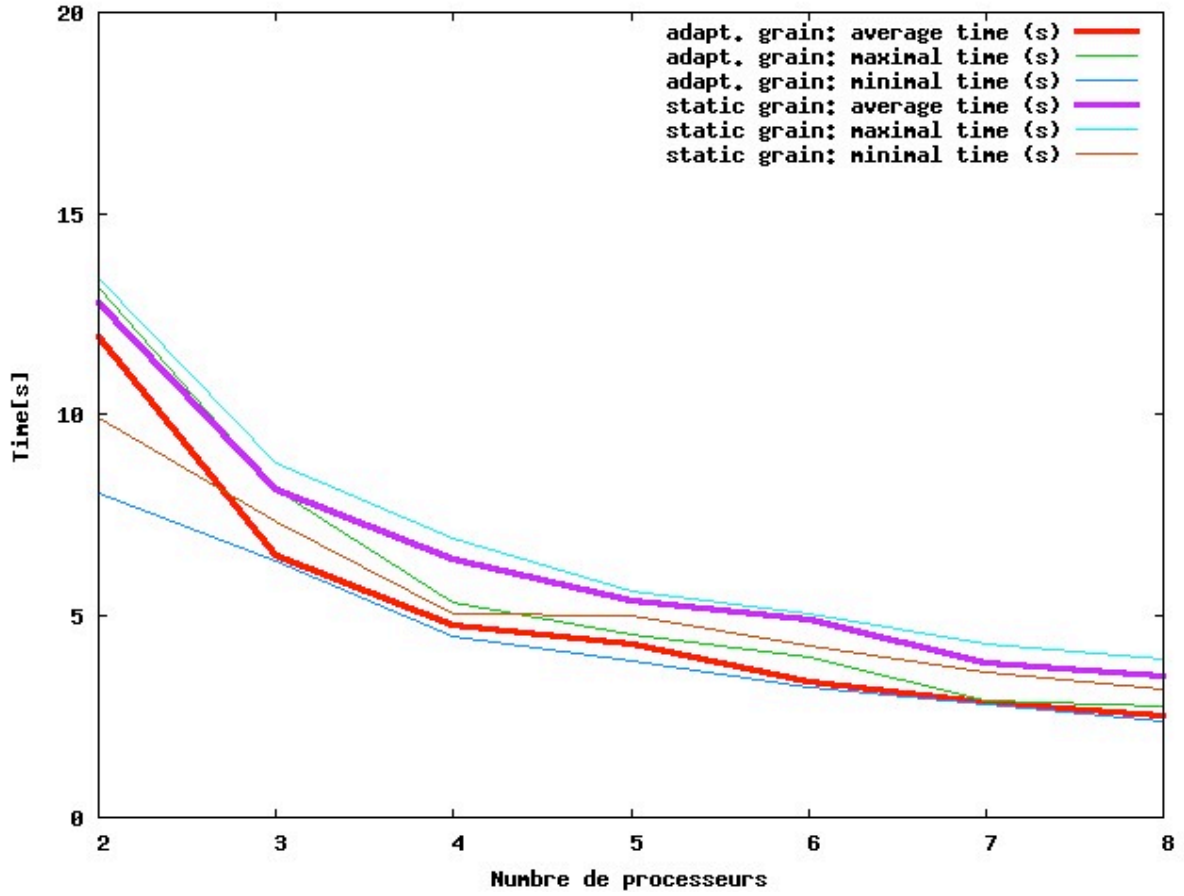
Adaptive algorithms

- Recursive computations
  - Local sequential computation
- Special case:
  - recursive extraction of parallelism when a resource becomes idle
  - But local execution of a sequential algorithm
- Example : prefix computation
  - Sequential : n operations
  - Parallel on p identical resources : at least 2n.(p/(p+1)) operations
  - Adaptive with work-stealing :
    - Coupling sequential and parallel partial-prefix computation
    - May benefit of an unbounded number or ressources
    - Performance : on p processors of variable speeds :2n/(p+1) + O(log n)

Adaptive Prefix versus optimal on identical processors



Adaptive Prefix with variable speeds



The race: sequential/parallel fixed/ Adaptive Prefix

	Sequentiel	Statique					Adaptatif
		p=2	p=4	p=6	p=7	p=8	p=8
Minimum	21,83	18,16	15,89	14,99	13,92	12,51	8,76
Maximum	23,34	20,73	17,66	16,51	15,73	14,43	12,70
Moyenne	22,57	19,50	17,10	15,58	14,84	13,17	11,14
Mediane	22,58	19,64	17,38	15,57	14,63	13,11	11,01

Tableau 1. *Compraison des temps des différents algorithmes lancés simultanément. Sur les 10 exécutions du test, l'algorithme adaptatif est le plus rapide.*

Questions?

[99] Samir Jafar, Thierry Gautier, Axel W. Krings, and Jean-Louis Roch. A checkpoint/recovery model for heterogeneous dataflow computations using work-stealing. EUROPAR'2005, Lisbonne, August 2005.
[97] Axel W. Krings, Jean-Louis Roch, and Samir Jafar. Certification of large distributed computations with task dependencies in hostile environments. IEEE EIT 2005, Lincoln, May 2005.
[92] Sébastien Varrette, Jean-Louis Roch, and Franck Leprévost. Flowcert: Probabilistic certification for peer-to-peer computations. IEEE SBAC-PAD 2004, pages 108-115, Foz do Iguacu, Brazil, October 2004.
[89] Samir Jafar, Varrette Sébastien, and Jean-Louis Roch. Using data-flow analysis for resilience and result checking in peer-to-peer computations. In IEEE DEXA'2004, Zaragoza, August 2004.