Result Certification Against Massive Attacks in Distributed Computations

Sébastien Varrette[1], Jean-Louis Roch[2] and Axel Krings[3]

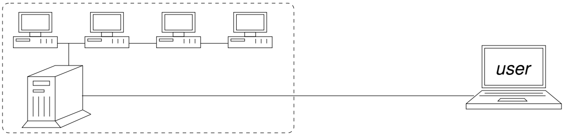
[1] Computer Science and Commnications Unit, University of Luxembourg, Luxembourg
[2] MOAIS team, LIG Laboratory, Grenoble, France
[3] Department of Computer Science, University of Idaho, Moscow, USA

Re-Trust 2008, Trento (Italy), October 15st, 2008

Summary

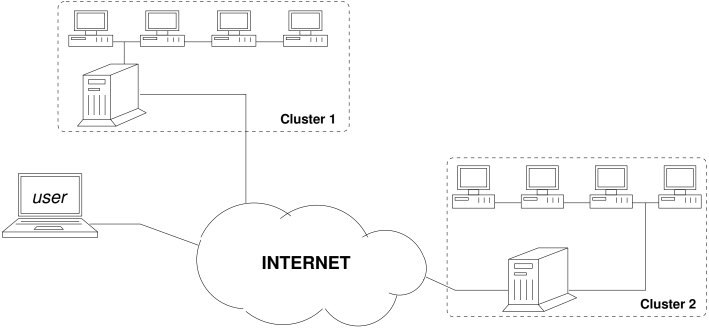
Large scale computing platforms

- [Beowulf] Clusters : Chaos.lu



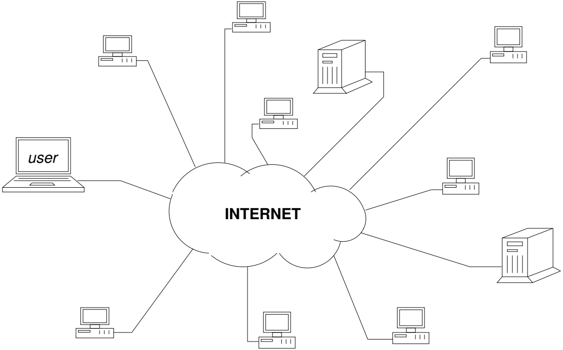
Large scale computing platforms

- Computing grids [Foster&al.97] : Grid5000, Globus etc.



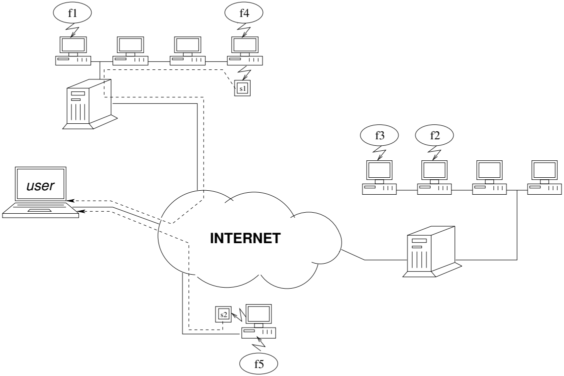
Large scale computing platforms

- « Desktop grid » : Seti@Home, BOINC, XtremWeb etc.



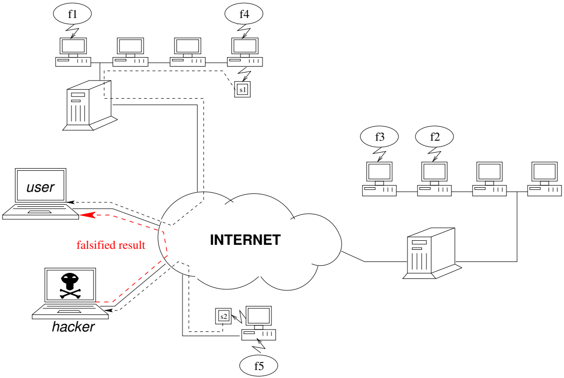
Result-Checking issue

- Falsified result : malicious act or not (cf. Seti@HOME [Molnar00])



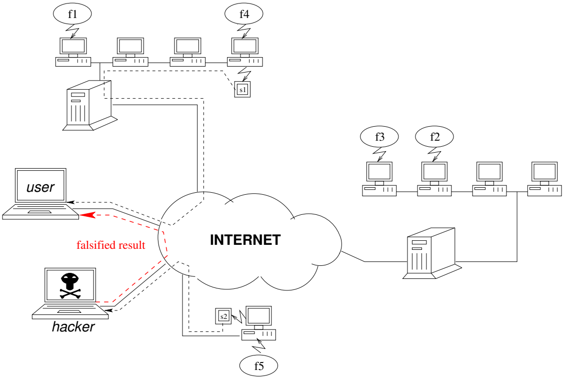
Result-Checking issue

- Falsified result : malicious act or not (cf. Seti@HOME [Molnar00])



Result-Checking issue

- Falsified result : malicious act or not (cf. Seti@HOME [Molnar00])



- Software Counter-measures : prevent before / **control after**

State of the art

- Essentially devoted to batchs of **independent** tasks

Specific approach : Simple checker [Blum97]

- check a [cheap] post-condition over computed results

  $\hookrightarrow$ DLP avec $|G| = n$: $L_n\left[\frac{1}{3}, \left(\frac{64}{9}\right)^{\frac{1}{3}}\right]$ – Simple checker $\mathcal{O}(\log n)$

- **The most efficient approach...** if possible !

State of the art

- Essentially devoted to batchs of **independent** tasks

Specific approach : Simple checker [Blum97]

- check a [cheap] post-condition over computed results

  $\hookrightarrow$ DLP avec $|G| = n$: $L_n\left[\frac{1}{3}, \left(\frac{64}{9}\right)^{\frac{1}{3}}\right]$ – Simple checker $\mathcal{O}(\log n)$

- **The most efficient approach...** if possible !

General approach : duplication

- Direct certification of the batch with sequential tests [Germain-Playez03]
- Batch reinforcement [Sarmenta03]
- **In all case :** attackers modelisation

State of the art

- Essentially devoted to batchs of **independent** tasks

Specific approach : Simple checker [Blum97]

- check a [cheap] post-condition over computed results

  $\hookrightarrow$ DLP avec $|G| = n$: $L_n\left[\frac{1}{3}, \left(\frac{64}{9}\right)^{\frac{1}{3}}\right]$ – Simple checker $\mathcal{O}(\log n)$
- **The most efficient approach...** if possible !

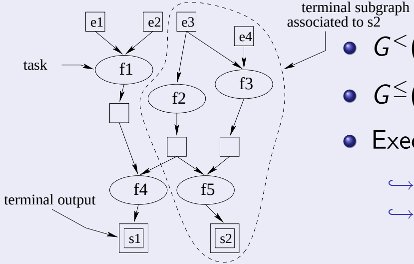
General approach : duplication

- Direct certification of the batch with sequential tests [Germain-Playez03]
- Batch reinforcement [Sarmenta03]
- **In all case :** attackers modelisation

$\Longrightarrow$ What about dependent tasks ?

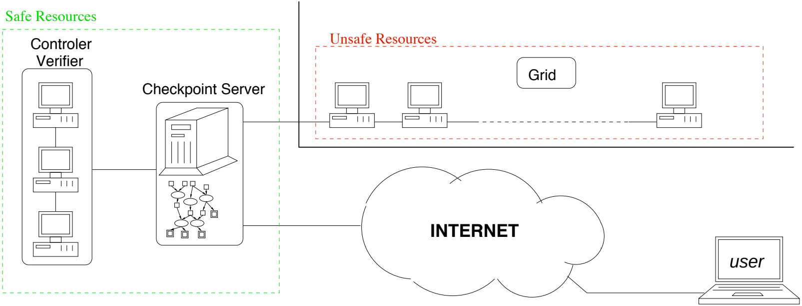
Execution model : macro-dataflow graph

Abstract representation of a parallel execution P(i)



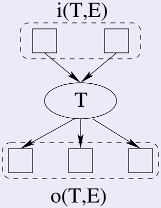
- $G^{<}(T)$: predecessors of T in G
- $G^{\leq}(T)$: $G^{<}(T) \cup \{T\}$
- Execution engine : KAAPI
  - $\hookrightarrow$ http://kaapi.gforge.inria.fr/
  - $\hookrightarrow$ C++ library for high performance parallel computing

Execution platform



- Resources partitionning $|R| \ll |U|$
- Reliable system for task re-execution

Impact of the faults (1)



- E : execution of $\mathbf{P}(i)$ over U resources
  - $\hookrightarrow$ G with intermediate values
  - $\hookrightarrow$ $T \in E : i(T,E) \longrightarrow o(T,E)$
- $\hat{E}$: execution of $\mathbf{P}(i)$ over R resources
  - $\hookrightarrow$ $T \in \hat{E} : \hat{i}(T,\hat{E}) \longrightarrow \hat{o}(T,\hat{E})$

Definition (execution state)

E is **correct** iff $E = \hat{E}$. Otherwise, E is **falsified**.

Task re-execution : compute $\hat{o}(T,E)$ from $i(T,E)$, compare to $o(T,E)$

Impact of the faults (2)

Definition (Correct and Faulty task)

- **Faulty task** T : $o(T, E) \neq \hat{o}(T, E)$

  $\hookrightarrow$ directly detected by controlers

  $\hookrightarrow$ **correct task** T : no task in $G^{\leq}(T)$ are faulty

- **Falsified result** : $o(T, E) \neq \hat{o}(T, \hat{E})$

  $\hookrightarrow$ hard to detect as $\hat{o}(T, \hat{E}) \neq \hat{o}(T, E)$

  $\hookrightarrow$ n_F falsified tasks

Monte-Carlo certification (1)

Definition (certification Monte-Carlo algorithm)

$$\mathcal{A} : (E, \varepsilon) \longrightarrow \begin{cases} \text{CORRECT (with error probability} \leq \varepsilon) \\ \text{FALSIFIED (with falsification proof)} \end{cases}$$

- Cf. Miller-Rabin
- **Interests** :
  - $\hookrightarrow$ ε fixed by the user
  - $\hookrightarrow$ a limited number of verifier calls (**ideally o(n)**)
  - $\hookrightarrow$ **can be done in parallel on R** !

Efficient detection of masive attack $(n_F \geq n_q = \lceil q.n \rceil)$

$\hookrightarrow$ the application **should** tolerate a limited number of faults

$\hookrightarrow$ no assumption on attackers behaviour except n_F

Monte-Carlo certification (2)

Resources	avg. speed/proc	total speed
U	Π_U	Π_U^{tot}
R	Π_R	Π_R^{tot}

- Scheduling by **on-line workstealing**
  - $\hookrightarrow$ execution (on U) : $\mathbf{W_1} \gg \mathbf{W}_\infty$
  - $\hookrightarrow$ certification (on R) : W_1^C and W_∞^C

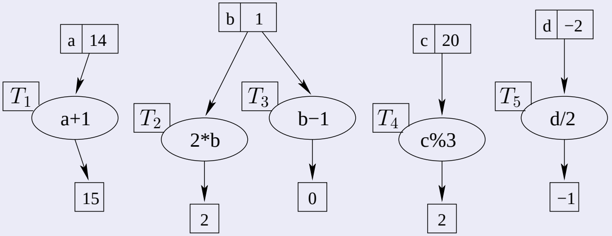
Theorem (Executing and Certification Time)

w.h.p :

$$T_{EC} \leq \left[\frac{W_1}{\Pi_U^{tot}} + \mathcal{O}\left(\frac{W_\infty}{\Pi_U}\right)\right] + \left[\frac{W_1^C}{\Pi_R^{tot}} + \mathcal{O}\left(\frac{W_\infty^C}{\Pi_R}\right)\right]$$

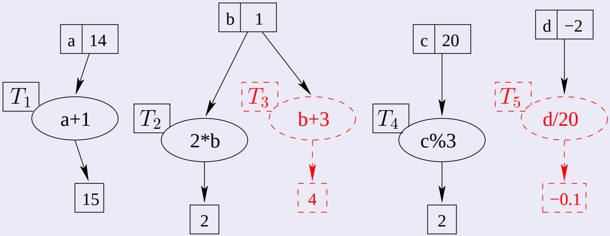
Independent case

Correct execution :



Independent case

Falsified execution :



Independent case

Monte-Carlo Test $MCT(E)$

Input: Execution E represented by G composed of independent tasks.
Output: The correctness of E (FALSIFIED or CORRECT)

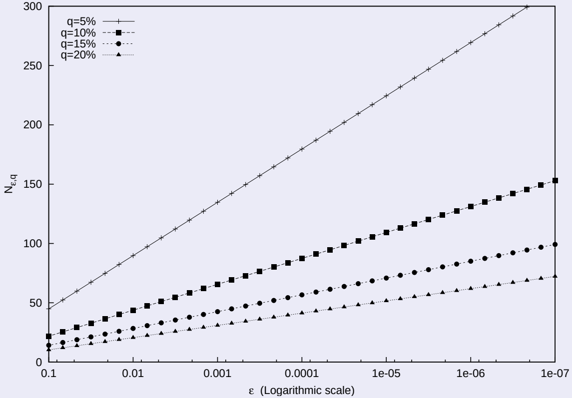
Uniformly choose one task T in G;
// Re-execution of T *on the* R *resources using the inputs* $i(T, E)$
$\hat{o}(T, E) \leftarrow$ `ReexecuteOnVerifier`$(\mathsf{T}, i(T, E))$;
if $o(T, E) \neq \hat{o}(T, E)$ **then**
    return FALSIFIED;
return CORRECT;

Theorem (Probabilistic certification by $MCT(E)$)

- $\mathcal{A}(E, \varepsilon)$: $N_{\varepsilon,q} = \lceil \frac{\log \varepsilon}{\log(1-q)} \rceil$ calls to $MCT(E)$
- $W_1^C \leq N_{\varepsilon,q} W_\infty$ and $W_\infty^C = W_\infty$
- $T_{EC} \leq \frac{W_1}{\Pi_U^{tot}} + \mathcal{O}\left(\frac{W_\infty}{\Pi_U}\right) + \frac{N_{\varepsilon,q} W_\infty}{\Pi_R^{tot}} + \mathcal{O}\left(\frac{W_\infty}{\Pi_R}\right)$

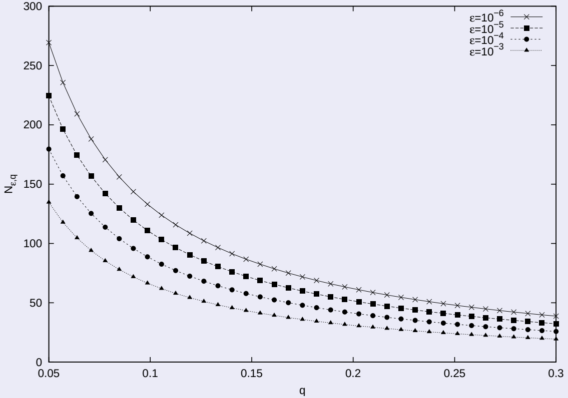
Independent case

Impact of ε over $N_{\varepsilon,q}$



Independent case

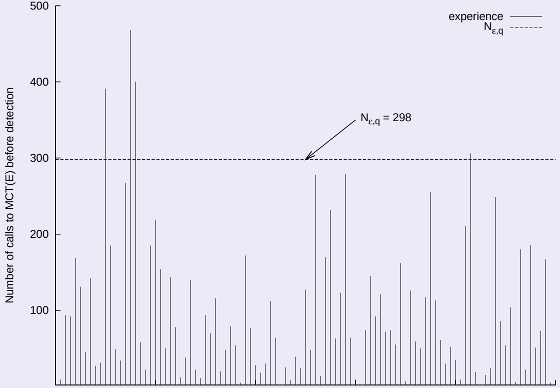
Impact of q over $N_{\varepsilon,q}$



Independent case

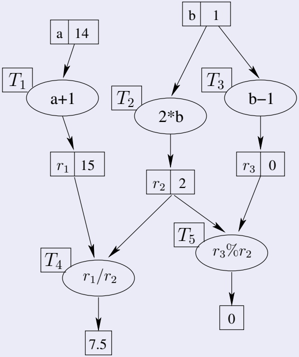
Non-detection illustration

$n = 10^6$, $q = 1\%$ and $\varepsilon = 5\%$



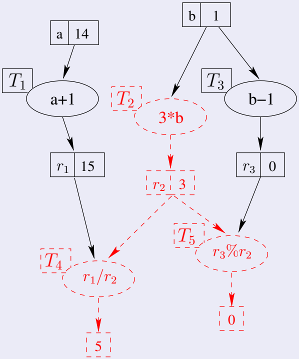
Dependent case

Correct execution :



Dependent case

Falsified execution :



Dependent case

- $\mathbf{n_I}$ **initiators** $\in \mathcal{I}(F) : \begin{cases} i(T,E) = \hat{i}(T,\hat{E}) \\ o(T,E) \neq \hat{o}(T,\hat{E}) \end{cases}$

  $\hookrightarrow$ falsified tasks you are **sure** to detect

- $\mathcal{P}\left(MCT(E) = CORRECT\right) \leq 1 - \frac{n_I}{n}$

Theorem (Minimal number of initiators)

For G with height h, maximal out-degree d and $n_F \geq n_q = \lceil q.n \rceil$

$$n_I \geq q \left\lceil \frac{n(d-1)}{d^h - 1} \right\rceil$$

Dependent case

Lemma (Initiators caracterization)

- $\mathcal{I}(F) = \{T_i \in F : F \cap G^{<}(T_i) = \emptyset\}$
- T is falsified $\Longleftrightarrow G^{\leq}(T) \cap \mathcal{I}(F) \neq \emptyset$

Extended Monte-Carlo Test $EMCT(E)$

Input: Execution E represented by G composed of dependent tasks.
Output: The correctness of E (FALSIFIED or CORRECT)

Uniformly choose one task T in G;
// Re-execution of $G^{\leq}(T)$ on R to detect initiators
forall $T_j \in G^{\leq}(T)$ / T_j *as not yet been checked* **do**
  $\hat{o}(T_j, E) \leftarrow$ `ReexecuteOnVerifier`$(T_j, i(T_j, E))$;
  if $o(T_j, E) \neq \hat{o}(T_j, E)$ **then**
    return FALSIFIED;
end
return CORRECT;

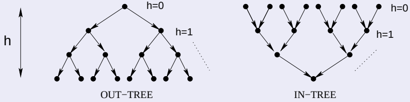
Dependent case

Theorem (Probabilistic certification by $EMCT(E)$)

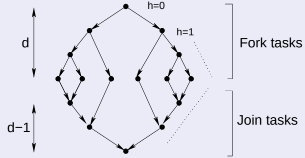
- $\mathcal{A}(E, \varepsilon)$: $N_{\varepsilon,q} = \lceil \frac{\log \varepsilon}{\log(1-q)} \rceil$ calls to $EMCT(E)$
- Expected cost per call : $C_G = \frac{1}{n} \sum_{T \in G} |G^{\leq}(T)|$
- **Worst case** : $W_1^C = \Omega(W_1)$ and $W_\infty^C = \Omega(W_\infty)$

Dependent case on some specific graphs

Trees



Fork-Join graphs



Theorem (Trees and Fork-Join graphs certification)

For G a tree or a Fork-Join graph with height h :

- $C_G \leq h + 3$
- $T_{EC} \leq \frac{W_1}{\Pi_U^{tot}} + \mathcal{O}\left(\frac{W_\infty}{\Pi_U}\right) + \mathcal{O}\left(\frac{hW_\infty}{\Pi_R^{tot}}\right) + \mathcal{O}\left(\frac{W_\infty}{\Pi_R}\right)$

$EMCT(E)$ variants to limit worst case cost

1. $EMCT_\alpha(E)$: check a proportion α of $G^{\leq}(T)$
2. $EMCT^K(E)$: check $\min\left(K, |G^{\leq}(T)|\right)$ tasks in $G^{\leq}(T)$

Definition (Minimal number of initiators)

Let $k \leq n_F$ and $V \subset \mathcal{V}_t$.

- *minimum number of initiators* with respect to V and k :

$$\gamma_V(k) = \min |G^{\leq}(V) \cap \mathcal{I}(F)| \quad \text{for} \begin{cases} |F| \geq k \\ G^{\leq}(V) \cap \mathcal{I}(F) \neq \emptyset \end{cases}$$

- *minimal initiator ratio* : $\Gamma_V(k) = \frac{\gamma_V(k)}{|G^{\leq}(V)|}$.

Note : $n_q \leq n_F$ is the smallest number of falsified tasks
$\Longrightarrow \gamma_G(n_q)$ is the smallest n_I possible.

$EMCT_\alpha(E)$ variants to limit worst case cost

$EMCT_\alpha(E)$

Input: Execution E represented by G composed of dependent tasks.
Output: The correctness of E (FALSIFIED or CORRECT)

Uniformly choose one task T in G;
$n_\alpha \leftarrow \lceil \alpha |G^{\leq}(T)| \rceil$; *//number of tasks to re-execute*
Define $\mathcal{T}_\alpha \subset G^{\leq}(T)$ composed of n_α tasks uniformly chosen in $G^{\leq}(T)$;
// Re-execution of $\mathcal{T}_\alpha$ on R to detect initiators
forall $T_j \in \mathcal{T}_\alpha$ / T_j *as not yet been checked* **do**
    $\hat{o}(T_j, E) \leftarrow$ `ReexecuteOnVerifier`$(T_j, i(T_j, E))$;
    if $o(T_j, E) \neq \hat{o}(T_j, E)$ **then**
        return FALSIFIED;
end
return CORRECT;

$EMCT_\alpha(E)$ variants to limit worst case cost

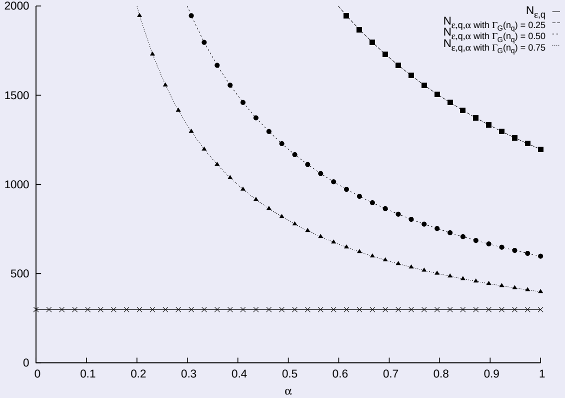
- Let $N_{\varepsilon,q,\alpha} = \begin{cases} \left\lceil \frac{\log \varepsilon}{\log(1-q\alpha\Gamma_G(n_q))} \right\rceil & \text{if } 0 < \alpha \leq 1 - \Gamma_G(n_q) \\ N_{\varepsilon,q} = \left\lceil \frac{\log \varepsilon}{\log(1-q)} \right\rceil & \text{otherwise.} \end{cases}$

Theorem (Probabilistic certification by $EMCT_\alpha(E)$)

- $\mathcal{A}(E,\varepsilon)$: $N_{\varepsilon,q,\alpha}$ calls to $EMCT_\alpha(E)$
- Expected cost per call : $C_G = \lceil \frac{\alpha}{n} \sum_{T \in G} |G^{\leq}(T)| \rceil$
- On average $W_1^C \leq \alpha \frac{N_{\varepsilon,q,\alpha}}{n} W_\infty \sum_{T \in G} \left| G^{\leq}(T) \right|$ and $W_\infty^C = \mathcal{O}(W_\infty)$

$EMCT_\alpha(E)$ variants to limit worst case cost

Impact of α on $\mathbb{N}_{\varepsilon,q,\alpha}$ (with $\varepsilon = 5\%$ and $q = 1\%$)



Certification algorithms comparison

Test $\mathcal{T}$ **:**		MCT §4	$EMCT$ §5.2	$EMCT_\alpha$ §5.3	$EMCT^1$ §5.4
$\#T$ detected faulty		$n_I \geq \left\lceil \frac{(d-1)n_F}{d^h-1} \right\rceil$	$n_q = \lceil n.q \rceil$	$n_q \alpha \Gamma_T(n_q)$ or n_q	$n_q \Gamma_T(n_q)$
$\mathcal{P}_{error}(\mathcal{T})$		$1-\Gamma_G(n_q) \leq 1-\left\lceil q\frac{(d-1)}{d^h-1} \right\rceil$	$1-q$	$1-q\alpha\Gamma_T(n_q)$ or $1-q$	$1-q\Gamma_T(n_q)$
$N^{\mathcal{T}}$: convergence to ϵ		$\left\lceil \frac{\log \epsilon}{\log(1-\Gamma_G(n_q))} \right\rceil$	$\left\lceil \frac{\log \epsilon}{\log(1-q)} \right\rceil$	$\left\lceil \frac{\log \epsilon}{\log(1-q\alpha\Gamma_G(n_q))} \right\rceil$ or $\left\lceil \frac{\log \epsilon}{\log(1-q)} \right\rceil$	$\left\lceil \frac{\log \epsilon}{\log(1-q\Gamma_G(n_q))} \right\rceil$
exact C_G		1	$\lvert G^{\leq}(T) \rvert$	$\lceil \alpha \lvert G^{\leq}(T) \rvert \rceil$	1
avg. C_G (n tasks, height h)	G	1	$\overline{\lvert G^{\leq} \rvert}$	$\left\lceil \alpha \overline{\lvert G^{\leq} \rvert} \right\rceil$	1
	Tree	1	$h+1 = \Theta(\log n)$	$\lceil \alpha(h+1) \rceil = \Theta(\alpha \log n)$	1
	Fork-Join	1	$h+3 = \Theta(\log n)$	$\lceil \alpha(h+3) \rceil = \Theta(\alpha \log n)$	1
W_1^C: $N^{\mathcal{T}}$ calls to $\mathcal{T}$	G	$N^{MCT} W_\infty$	$N^{\mathcal{T}} W_\infty \overline{\lvert G^{\leq} \rvert}$	$\alpha N^{\mathcal{T}} W_\infty \overline{\lvert G^{\leq} \rvert}$	$N^{EMCT^1} W_\infty$
	Tree	$N^{MCT} W_\infty$	$\mathcal{O}(hW_\infty)$	$\mathcal{O}(\alpha h W_\infty)$	$N^{EMCT^1} W_\infty$
	Fork-Join	$N^{MCT} W_\infty$	$\mathcal{O}(hW_\infty)$	$\mathcal{O}(\alpha h W_\infty)$	$N^{EMCT^1} W_\infty$
W_∞^C		$\mathcal{O}(W_\infty)$	$\mathcal{O}(W_\infty)$	$\mathcal{O}(W_\infty)$	$\mathcal{O}(W_\infty)$

Conclusion

Result-checking for distributed computations

- Approach based on macro-dataflow analysis
  - $\hookrightarrow$ deals with task dependencies
- "No" hypothesis on attacker behaviour
- Monte-carlo certification [E]MCT[X]
  - $\hookrightarrow$ low overhead for recursive/Fork-Join programs
  - $\hookrightarrow$ high overhead in general ($\longrightarrow$ $EMCT_\alpha(E)$ and $EMCT^K(E)$)
  - $\hookrightarrow$ validation on medical application (not presented here)

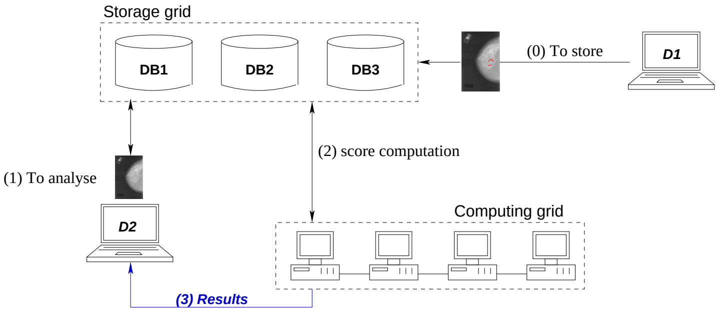
Perspective/Current work

- Atlantic city extension
  - $\hookrightarrow$ verifier not so accurate
  - $\hookrightarrow$ if test fails, probability to stand below the tolerance threshold ?
- Dealing with $n_F < \lceil n.q \rceil$
  - $\hookrightarrow$ Algorithm-Based Fault-Tolerance (ABFT)

Thanks for your attention...

Questions ?

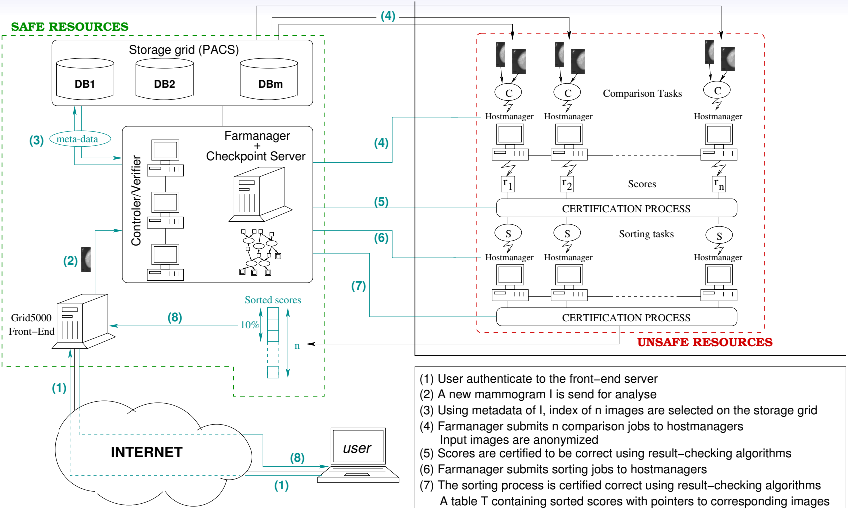
Proof of concept



Breast cancer lesions detection in mammogrames [Varrette& al.06]

- statistical comparison on a database of studied cases

Experimental protocol

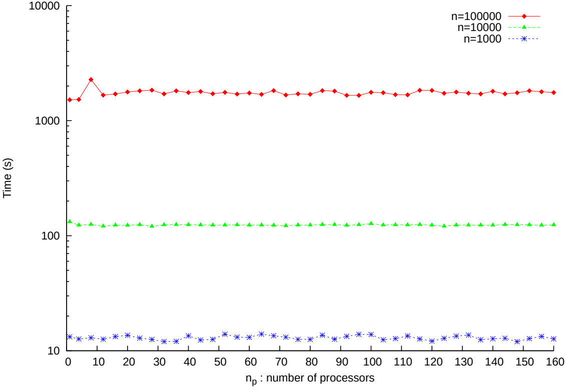


(1) User authenticate to the front−end server
(2) A new mammogram I is send for analyse
(3) Using metadata of I, index of n images are selected on the storage grid
(4) Farmanager submits n comparison jobs to hostmanagers
Input images are anonymized
(5) Scores are certified to be correct using result−checking algorithms
(6) Farmanager submits sorting jobs to hostmanagers
(7) The sorting process is certified correct using result−checking algorithms
A table T containing sorted scores with pointers to corresponding images is produced
(8) The first 10% entries of T are sent back to the user

Experimentations

Deployment on Grid5000 ; $\varepsilon = 0.001$, $q = 0.01$ ($N_{\varepsilon,q} = 688$)

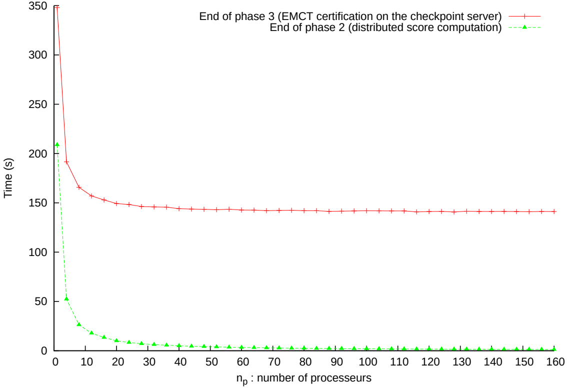
Time required to deploy the images on the grid



Experimentations

Deployment on Grid5000 ; $\varepsilon = 0.001$, $q = 0.01$ ($N_{\varepsilon,q} = 688$)

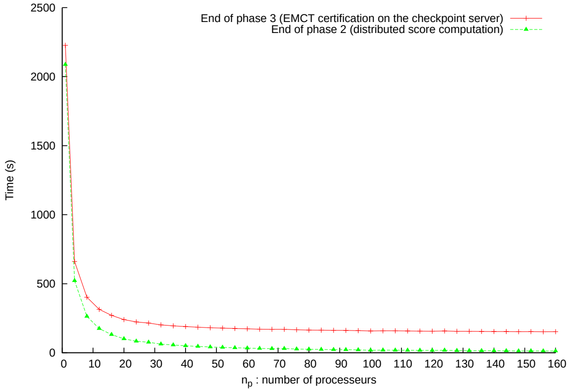
Scores computation + certification : 1000 tasks



Experimentations

Deployment on Grid5000 ; $\varepsilon = 0.001$, $q = 0.01$ ($N_{\varepsilon,q} = 688$)

Scores computation + certification : 10000 tasks



Experimentations

Deployment on Grid5000 ; $\varepsilon = 0.001$, $q = 0.01$ ($N_{\varepsilon,q} = 688$)

Scores computation + certification : 100000 tasks