Probabilistic Certification of Divide & Conquer Algorithms on Global Computing Platforms.

Application to Fault-Tolerant Exact Matrix-Vector Product

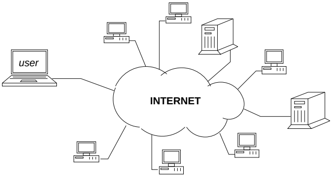
Jean-Louis Roch – Sébastien Varrette

MOAIS Team, INRIA and LIG, Grenoble, France Institut National Polytechnique de Grenoble, INPG, France
Universit du Luxembourg, LACS, Luxembourg

Outlines

Computation power of Grid & P2P platform

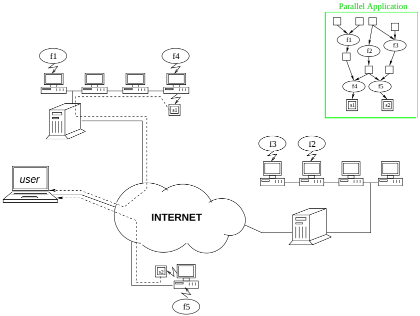
- Compute intensive applications with potential parallelism
- Global computing platforms (Grid, P2P) : offer an "unbound" computation power



  - *Volunteer Computing* : steal idle cycles through the Internet
  - Seti@Home : 900000 machines $\Longrightarrow$ 250 Tflops
  - BOINC/`Folding@home` : 650 TFlops in 2006 (incl. PS3)
  - Sharcnet (Ontario), Grid5000 (France), DAS (Netherland), ...
- yet *unbound environment* subject to attacks

Global computing architecture

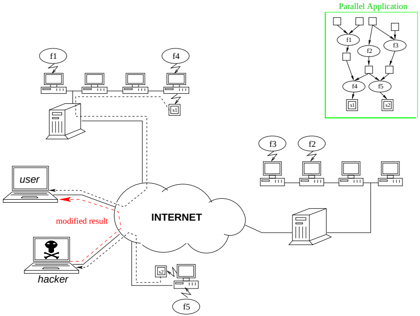
- Grid and P2P : Transparent allocation of the resources to authenticated users
  scheduling supports ressource connections / disconnections



Global computing architecture and task forgery

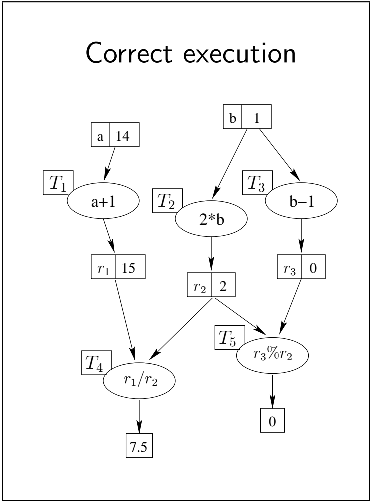
- Grid and P2P : Transparent allocation of the resources to authenticated users

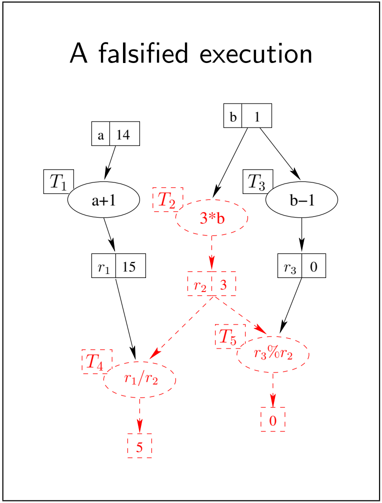
  scheduling supports ressource connections / disconnections



- Yet a task can be **forged** $\iff f(\text{ input }) \neq \hat{f}\ (\text{ input })$
- forgery can affect *many* tasks [e.g. patched client in SETI@home]

Task forgery and result falsification





State-of-the-Art in Result Certification

- Mainly target programs **P** composed of independent tasks
- Specific approach : *check post-condition* on the results
  - Eg : Sorting $\mathcal{O}(n \log n)$ – Simple-Checker $\mathcal{O}(n)$ [Blum97]
  - **The most efficient approach when possible !**
- General approach : *Replication-based*
  - Voting [*e.g.* BOINC, SETI@home]
  - Spot-checking [Germain-Playez03, based on Wald test]
  - Blacklisting, Credibility-based fault-tolerance [Sarmenta03]
  - Partial execution on reliable resources [Gao-Malewicz04]

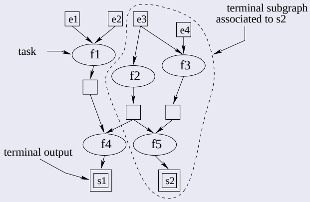
Yet, no guarantee of result correction without hypothesis on the attack

Approach : Massive versus localized attacks

- Practical global computing framework :
  - DAG of tasks, highly parallel
  - for most executions, only few or none forged tasks !
    $\hookrightarrow$ full replication useless and too expensive

- Yet, **no blind trust** :
  - few falsifications are possible $\hookrightarrow$ can be efficiently corrected by Algorithm-based fault tolerance (**ABFT**)
    [Beckman 93, Plank&al 97, Saha 2006]
  - large scale falsifications are possible $\hookrightarrow$ can be detected by trustable verification of randomly selected tasks :
    Extended Monte-Carlo Certification **EMCT**
    [Krings&al 06]

Abstract representation of the execution of $\mathbf{P}(\overrightarrow{x})$



- Bipartite DAG $G = (\mathcal{V}, \mathcal{E})$.
  1. Tasks T_j
  2. Data (inputs and outputs)
- Hyp : G is deterministic

Some useful notations :

- $n = |\{T_j \in G\}|$
- $G^{<}(T)$: sub-graph induced by predecessors of T in G
- $\mathbf{G}^{\leq}(\mathbf{T}) = G^{<}(T) \cup \{T\}$

EMCT : Monte-Carlo detection of massive attack

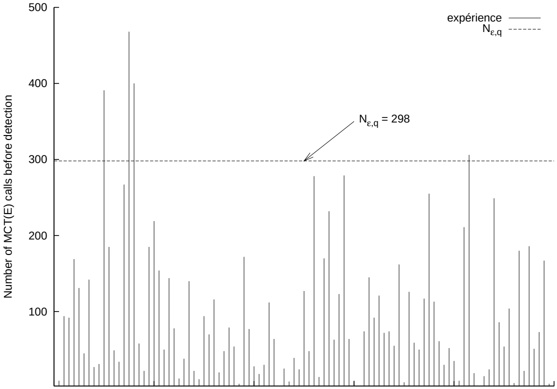
EMCT : Monte-Carlo detection of massive attack

Input : G, an execution E composed of dependent tasks
Output : the correctness of E (CORRECT or FALSIFIED)

Pick up randomly $T \in G$;
// Verify if $G^{\leq}(T)$ contains no faulty tasks
forall $T_j \in G^{\leq}(T)$ / T_j *has not yet been checked* **do**
    $\hat{o}(T_j, E) \leftarrow$ `ReexecuteOnControler` T_j, $i(T_j, E)$;
    if $o(T_j, E) \neq \hat{o}(T_j, E)$ **then return** FALSIFIED ;
end
return CORRECT ;

- if #falsified tasks $\geq q.n$ then $N_{\epsilon,q} = \lceil \frac{\log \epsilon}{\log(1-q)} \rceil$ calls to $EMCT(E)$ ensures error probability $\leq \epsilon$

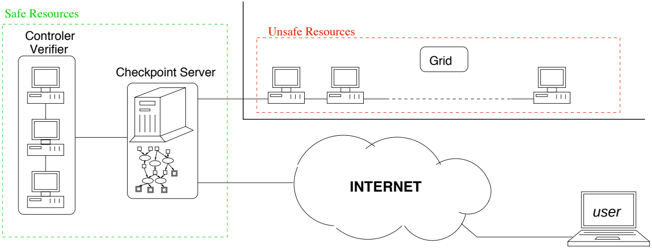
Example : $n = 10^6$ tasks, falsification rate $q = 1\%$
$\hookrightarrow$ 298 calls to $EMCT(E)$ ensures detection of the attack with error probability 5%



EMCT Execution and certification platform

- Execution engine able to use/generate G
  $\hookrightarrow$ eg *Kaapi*, XtremWeb, etc.
- **Checkpoint server** : stores the tasks graph G
- **Controllers** : extract $T \in G$ & re-execute $G^{\leq}(T)$ in a trusted way

$\Longrightarrow$ Partition GC resources into $\begin{cases} U \text{ (unreliable)} \\ R \text{ (reliable)} \end{cases}$ $\quad |R| \ll |U|$



EMCT interest for certification of global computations

Interest

- EMCT : avoids strong replication to detect *massive attack* with ratio $\geq q$ w.h.p.
- ϵ fixed by the user
- a limited number of task re-execution per call

Purpose

- complete certification with only few task verifications on reliable resources
- Cost of one call to EMCT : $|G^{\leq}(T)|$ task to re-compute (potentially high in the worst case)
- **Expected cost** $= C_G = \frac{1}{n} \sum_{T \in G} |G^{\leq}(T)|$
  proved small when G is a fork-join graph [theorem 1]

On the interest of using GC platforms...

Resource Type	avg. speed/proc	total speed	Usage
U	Π_U	Π_U^{tot}	execute E
R	Π_R	Π_R^{tot}	certification

Speed : number of unit operations per second

Bound on T_{EC} (time required for execution+certification)

Based on work-stealing [Bender-Rabin02] , with high probability

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

But, in the worst case, $W_1^C = \Omega(W_1)$ and $W_\infty^C = \Omega(W_\infty)$.

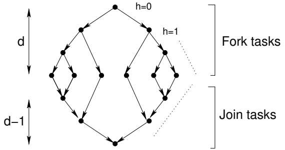
W_1 : total work for execution
W_∞ : depth work for execution

W_1^C : total work for certif.
W_∞^C : depth work certif.

Definition : **Fork-Join** graph : G is either :

- a graph with only one vertex (both source and sink) ;
- the parallel composition of $k > 0$ Fork-Join graphs $G_1, \ldots, G_k$

(parallel Divide&Conquer programs)



Theorem 1 : $EMCT(E)$ expected cost on Fork-Join graphs

If G is either a tree or a Fork-Join graph, the expected number of tasks to re-execute in one $EMCT(E)$ call is $C_G \leq h + 3$.
In addition,

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

$\hookrightarrow$ low overhead on R if $W_\infty \ll W_1$

Application to fault-tolerant global computation

Certification of Fork-Join computation

- $T_{EC} \leq \frac{W_1}{\Pi_U^{tot}} + \mathcal{O}\left(\frac{W_\infty}{\Pi_U}\right) + \mathcal{O}\left(\frac{hN_{\epsilon,q}W_\infty}{\Pi_R^{tot}}\right) + \mathcal{O}\left(\frac{W_\infty}{\Pi_R}\right).$
- if $W_\infty \ll W_1$: very low average overhead

Enable the design of fault-tolerant computation

Application to certified matrix-vector product iteration :

- Given $k \times k$ matrix A, compute $z_i := x_i A$ for many x_i
- Each product $z_i := x_i A$: *fork-join* with $W_\infty = O(\log k) \ll k^2$
  $\Longrightarrow$ efficient detection of a massive attack

yet requires ABFT matrix-vector product to tolerate error rates $< q$

ABFT matrix-vector with low rate error correction $< q$

- some schemes proposed for exact computations in rings [Beckman 1993, Saha 2006]
- for linear algebra based on parity checkpointing [Plank 1997, Dongarra 2006]

Use of a linear BCH error-correcting code

- Choose a cyclic code (n, k, d) on $\mathbb{F}$ with distance $d = \Omega(k)$: let G a $k \times n$ generator matrix.
  Precompute on the reliable resources : $B := A.G$ in $O(k^2 \log k)$.
- Let $q = \frac{d-1}{2n}$: then $z_i = x_i B$ tolerates $< q.n$ errors on z_i
- E.g. using a MDS code with distance $d = \frac{k}{2} + 1$:
  then $n = \frac{3k}{2}$ and $q = \frac{1}{6}$.

Certified vector-matrix computation : $y_i = x_i.A$

Computation on the Unreliable global computing resources U

- for each vector-matrix product : **Compute** $z_i := x_i.B$
  Cost : $W_1 = \mathcal{O}(kn) = \frac{1}{1-2q} W_{seq}$ and $W_\infty = \mathcal{O}(\log k)$
- NB : if no falsification occurs, $z = yG$. *Moreover, y can be recovered if less than $\frac{d-1}{2}$ components of z_i are incorrect.*

Certification/correction on the Reliable resources R

- **Use EMCT to detect** if more than $q.n = \frac{d-1}{2}$ tasks have been forged, by testing $N_{\epsilon,q} = \frac{\log \epsilon}{\log 1-q}$ random tasks :
  Expected cost : $W_1 = N_{\epsilon,q}\mathcal{O}(\log^2 k)$ and $W_\infty = \mathcal{O}(\log k)$
- If forgery detected : restart computation of z.
- Else, **recover y using BCH decoding of z**
  Cost : $W_1' = \mathcal{O}(\frac{k \log k}{1-2q})$ and $W_\infty' = \mathcal{O}\left(\log^2\left(\frac{k}{1-2q}\right)\right)$.

Computation and certification time

- On Unreliable resources, whole time T_U :

$$T_U = \mathcal{O}\left(\frac{k^2}{(1-2q)\Pi_U^{tot}} + \frac{\log k}{\Pi_U}\right) \qquad \simeq \frac{3}{2}\frac{W_{seq}}{\Pi_U^{tot}}$$

$\hookrightarrow$ corresponds to the arithmetic cost

- On Reliable resources, whole time T_R :

$$T_R = \mathcal{O}\left(\frac{k \log k}{(1-2q)\Pi_R^{tot}} + \frac{\log^2 k}{\Pi_R}\right)$$

$\hookrightarrow$ cost dominated by BCH decoding, yet not by EMCT.

- However, precomputation of $B = AG$ in $O(k^2 \log k)$: amortized with $\gg \log k$ vector-matrix products

Conclusion

Summary

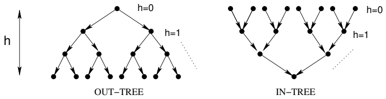
- EMCT certification against massive attacks :
  small expected cost on reliable resources for fork-join graph
- coupled to ATBF an algorithm, enables certification of results with no assumption on the attacks

Perspectives

- other (linear algebra) computations with dependencies
- improving underlying ATBF algorithms :
  numerical computation ; sparse matrix A ;

Questions ?

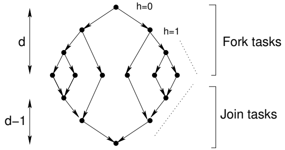
Expected Efficiency of EMCT for tree programs



Number of tasks to be verified on R in one call to EMCT :

- for out-tree $=$ h tasks at most
- for in-tree : worst case $=$ n tasks
  but *expected number of tasks* $= \frac{\sum_{T_i \in G} \left| G^{\leq}(T_i) \right|}{|G|} \leq h+1$
- $\Longrightarrow$ expected cost of one EMCT call : $O\left(W_\infty\right)$
  very low overhead if $W_\infty \ll W_1$

Extension to fork join parallel programs



- ForkJoin graph G :
  - may be unbalanced, but symmetric : $G = G_F \cup G_J$
- Expected cost (lemma 3) $\leq 2(d+1) = h+3$

$$\begin{array}{rcl} n.C_G & = & \sum_{T \in G} |G^{\leq}(T)| \\ & = & \sum_{T \in G_F} |G^{\leq}(T)| + \sum_{T' \in G_J} |G^{\leq}(T')| \\ & \leq & \sum_{T \in G_F} |G_F^{\leq}(T)| + \sum_{T' \in G_J} 2.|G_J^{\leq}(T')| + d + 1 \\ & \leq & \leq 2(d+1)n = (h+3)n \end{array}$$