Building Secure Resources to Ensure Safe Computations in Distributed and Potentially Corrupted Environments

Sébastien Varrette[1], Jean-Louis Roch[2], Guillaume Duc[3] and Ronan Keryell[3,4]

[1] Computer Science and Commnications Unit, University of Luxembourg, Luxembourg
[2] MOAIS team, LIG Laboratory, Grenoble, France
[3] HPCAS team, Computer Science Laboratory, TÉLÉCOM Bretagne, Plouzané, France
[4] HPC Project, Meudon, France

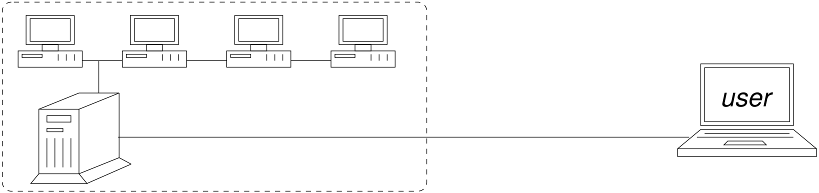
SGS 2008, Las Palmas de Gran Canaria, August 25st, 2008

Summary

Large scale computing platforms (I)

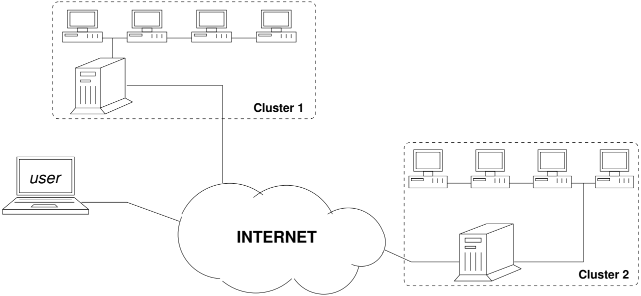
Highly demanding applications needs highly parallel computing infrastructures

- [Beowulf] Clusters: Chaos.lu (cluster @ Luxembourg)



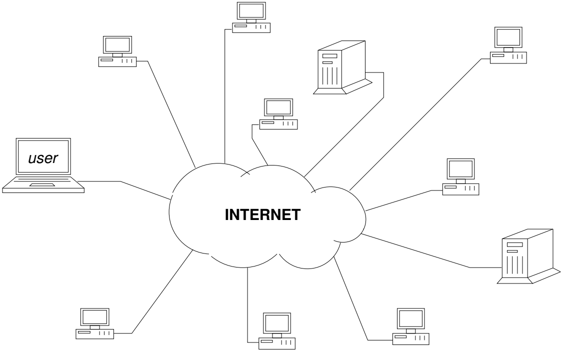
Large scale computing platforms (II)

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



Large scale computing platforms (III)

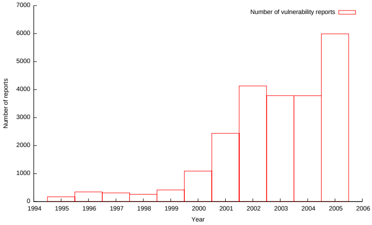
- "Desktop grid": Seti@Home, BOINC, XtremWeb, etc.

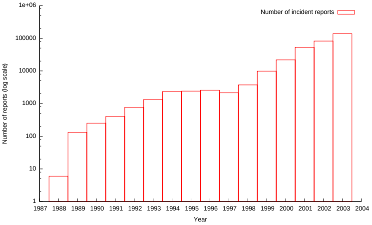


Threats...

Rather open infrastructures and public networks ⇝

- Scans, DoS, DDoS, intrusion
- Applicative vulnerabilities





- Malwares
  - worms, virus (need host program to replicate), trojan horses...
- The "Seti@Home" problem
  - In 2000, modified client to improve FFT computation but introduced rounding errors that canceled months of world-wide computation... ☹
  - A node can reply "not found" to keep a good result for her own

... And security concerns

General constraints: *CAIN + AD*

CONFIDENTIALITY
AUTHENTICATION
INTEGRITY
(NON-REPUDIATION)

+

AVAILABILITY
DELEGATION

- Availability for fault tolerance (crash-fault...)
- Delegation for access right

Specific constraints:

- Interaction between global/local security policies
- Single Sign On
- Rely on standards + scalability

...Trust scalability issue

Secure grid computing in a real (hostile) environment

- No confidence in the remote computers that run our own programs
- What proves the remote computers are reliable and trustworthy?
- The remote administrator or a pirate can spy computations
- The remote administrator or a pirate can modify computations and results

Distributed computing

⇝ Asymmetry in the trust from the user point of view

- A remote computer can trust a user with secure authentication
- ...but how to be sure the remote program is fairplay?
- The remote computer should be able to verify the policy usage

In this talk

- Guidelines for a secured large scale computing platform
  - ↪ ensure general/specific security concern
- Explicit construction of strongly secured resources
  - ↪ used to ensure computation resilience against tasks forgery
  - ↪ combine both software and hardware approaches
- Application within the SAFESCALE project

Summary

Guidelines for a secure computing grid (I)

Build Safe Resources

- Control user rights, limit available services, enforce quotas
- Ensure up-to-date system, enable firewall, monitoring and audit
- Sandboxing
- Hard drive encryption
- Anti-virus, etc.
- ... and more in the sequel

Guidelines for a secure computing grid (II)

Ensure confidentiality

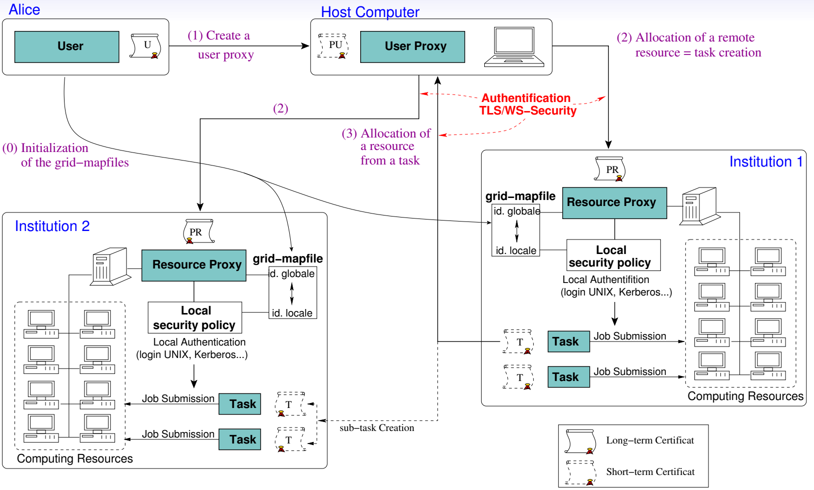
- Communications:
  - ↪ Cluster/grid of cluster: VPN, SSH, eventually IPSec...
  - ↪ "Globus" grids: SSL/TLS, WS-Security, WS-SecureConversation
- [Source | Executed] code
  - ↪ encrypted computation
  - ↪ code obfuscation
  - ↪ time-limited blackbox security

Guidelines for a secure computing grid (III)

Ensure authentication & (eventually) access control

- Clusters: SSH + authentication agents, Kerberos, KryptoKnight, LDAP(s)-based
- Globus: GSI (Grid Security Infrastructure) module

Guidelines for a secure computing grid (IV)



Guidelines for a secure computing grid (V)

Ensure integrity

- Communications: Modification Detection Code, Message Authentication Code, etc.
- Parallel execution resilience against crash-faults/task forgery
  - ↪ based on macro-dataflow graph analysis
  - ↪ graph stored on a secure checkpoint server for checkpoint/rollback
  - ↪ task context extracted for safe re-execution and result checking
  - ↪ assume partition of the resources (reliable ∪ unreliable)

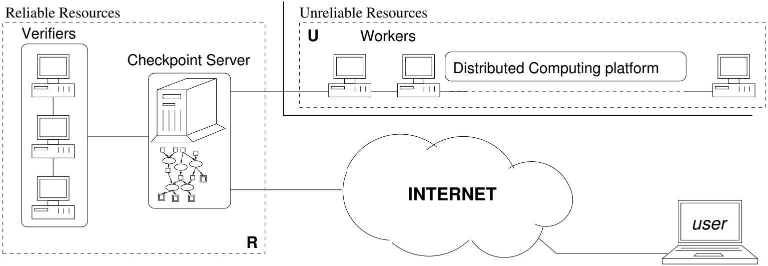
Guidelines for a secure computing grid (VI)

Monte-Carlo certification by partial duplication [Varrette07]

- Efficient certification of independent tasks: $MCT(E)$
- Certification of dependent tasks
  - ↪ $EMCT(E)$: low-overhead certification for Trees/Fork-Join graphs
  - ↪ $EMCT(E)$ variants to limit worst case cost:

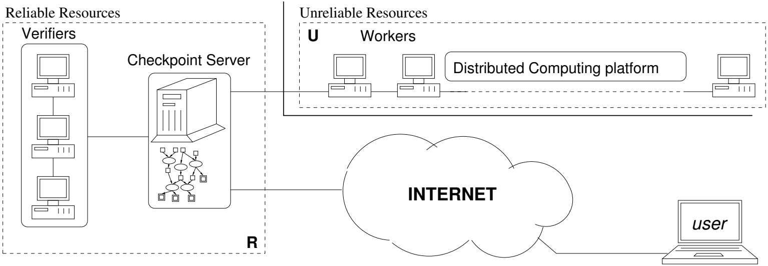
$$EMCT_{\alpha}(E),\ EMCT^{K}(E)$$

⤳ Execution platform in SAFESCALE for safe execution



- Resources partitionning $|Reliable| \ll |Unreliable|$
- Reliable system for task re-execution
- R need to be trusted...

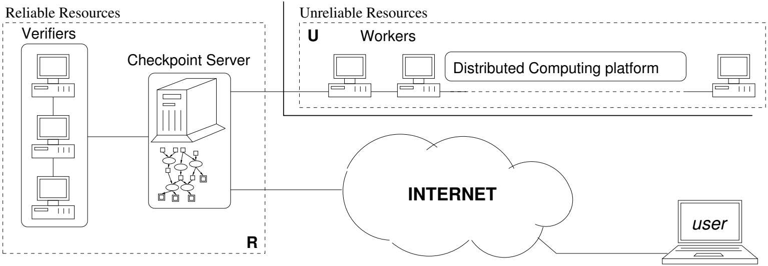
⤳ Execution platform in SAFESCALE for safe execution



- Resources partitionning $|Reliable| \ll |Unreliable|$
- Reliable system for task re-execution
- R need to be trusted...

$\Longrightarrow$ Effective construction of strongly secured resources?

⇝ Execution platform in SAFESCALE for safe execution



- Resources partitionning $|Reliable| \ll |Unreliable|$
- Reliable system for task re-execution
- R need to be trusted...

$\Longrightarrow$ Effective construction of strongly secured resources?

Hybrid solution: software + **hardware**

Software environment

- Programming model $\equiv$ Task oriented parallelism to cope with SAFESCALE model
- KAAPI C++ framework (TBB-like language) developed at LIG to express task parallelism and work stealing
  - Task creation
  - Shared types to hide communications if needed
  - Parallel iterators
- Current development of an automatic parallelizer based on PIPS source-to-source compiler
  - Use directives to delimit task creation
  - Use PIPS semantics analysis to parallelize the code
  - Use of array region analysis to compute data to be changed into shared object

`http://www.cri.ensmp.fr/pips`

Summary

Needs for some hardware support

- The verifiers must be trusted...
- A trusted and secure architecture may be used for computation without verification
- A node may want to verify what alien program is running
  - Is the usage contract respected?
  - Does the binary correspond to a given program or even source?
- Difficult to hide secrets into binaries against reverse-engineering

$\rightsquigarrow$ Useful to have some secure hardware too...

Some definitions (I)

- About what we want to protect into a secure processor

Definition

A secure process

- Is protected against physical action outside
- Is protected against logical action inside
- Has memory spaces enciphered outside
- Has a partially randomized address space

Definition

A secure execution of a secure process is

- Correct (no attack on its states detected up to now...)
- Or aborted (active attack detected and all the internal states are deleted)

Some definitions (II)

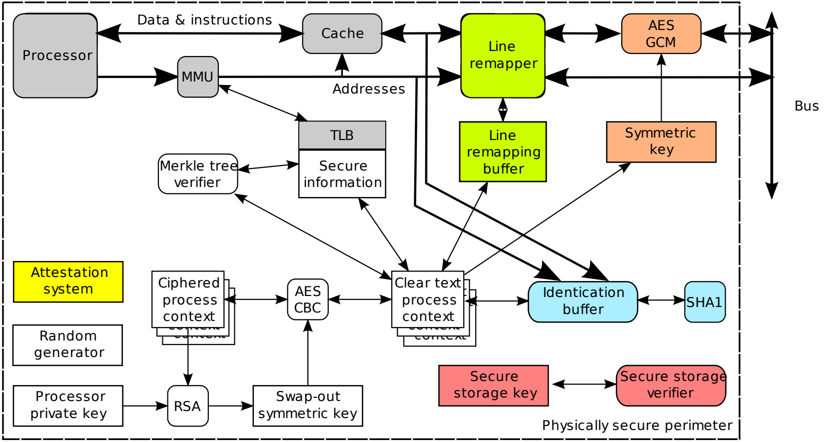
- About the attackers

Definition

An attacker of a secure process is

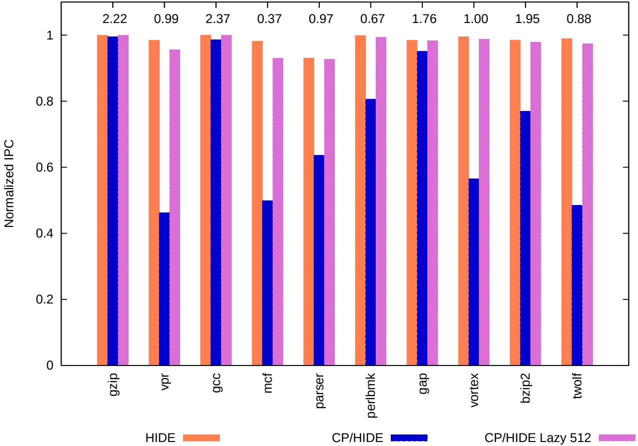
- Another process (secure or not, the operating system...) that spies or modifies internal states (registers, caches...) or external states (memory, peripherals...)
- A human being with logical or physical means to forge or spy anything outside the processor

CryptoPage: the big picture



Performance simulations on SpecINT2000

On SimpleScalar/CryptoPage



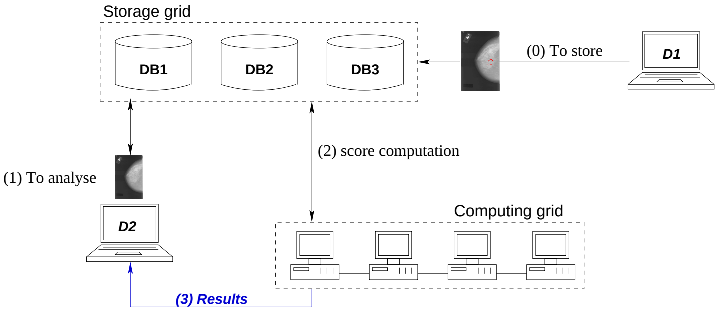
CryptoPage use case

To run a secure process remotely

- The compute owner enciphers her program by using the public key of the remote processor
- The remote processor executes the process
- The remote owner can authenticate the process against a given binary or a given source with a a given compilation chain

Summary

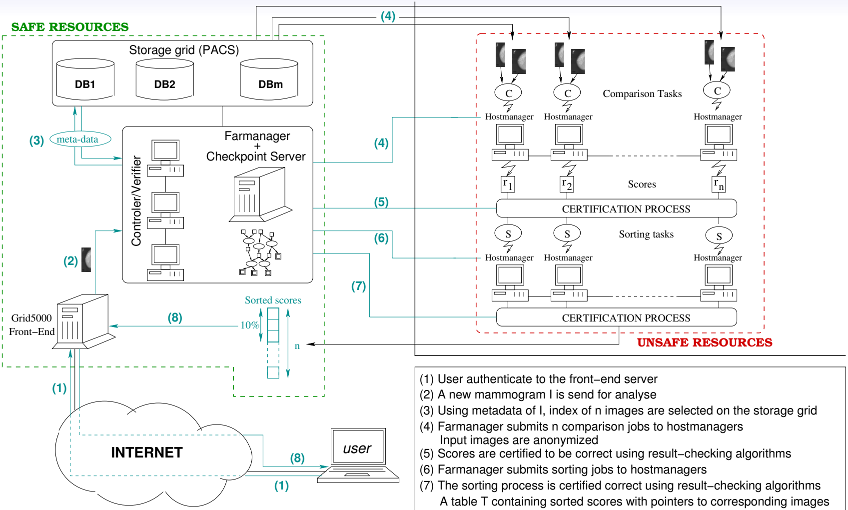
SAFESCALE application



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

- Statistical comparison on a database of studied cases

Experimental protocol



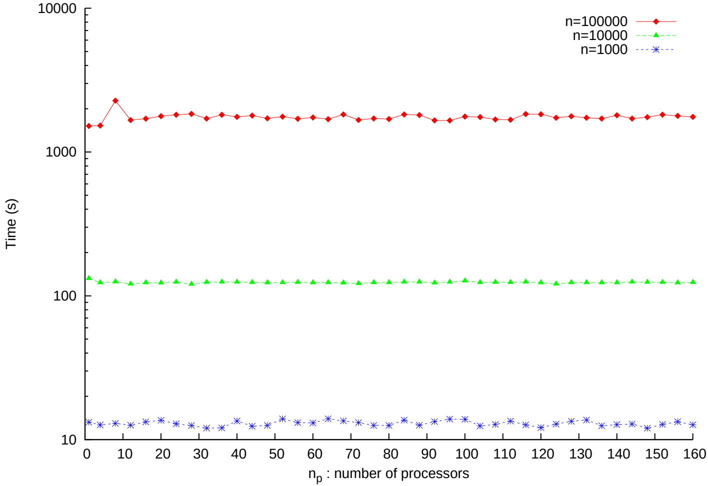
(1) User authenticate to the front–end server
(2) A new mammogram I is send for analyse
(3) Using metadata of I, index of n images are selected on the storage grid
(4) Farmanager submits n comparison jobs to hostmanagers
Input images are anonymized
(5) Scores are certified to be correct using result–checking 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

Experimental results (I)

- Try to detect corruption with ratio of wrong nodes $q = 0.01$ with a probability of $\varepsilon = 0.001$
- With only 1 reliable processor to do the verification of 688 tasks needed by *EMCT*
- The execution on CryptoPage is estimated with an overhead of 7.4% (worst case on SpecINT 2000)
- The data-base access is not yet parallelized

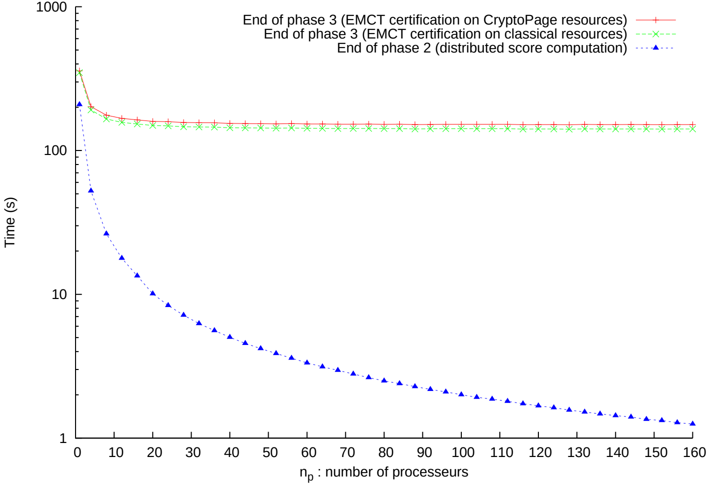
Experimental results (II)

Time required to deploy the images on the grid



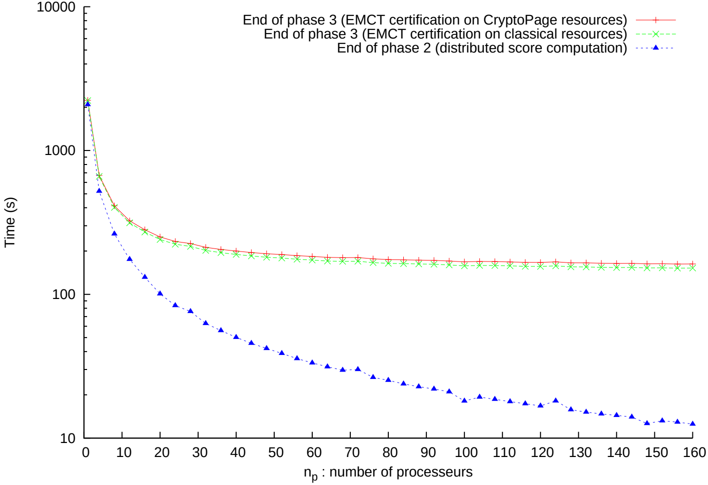
Experimental results (III)

Scores computation + certification: 1000 tasks



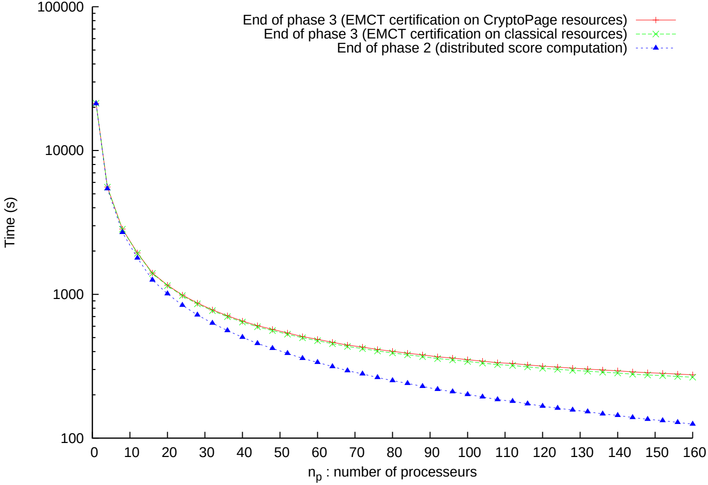
Experimental results (IV)

Scores computation + certification: 10000 tasks



Experimental results (V)

Scores computation + certification: 100000 tasks



Conclusion

- Security, reliability and trust need to be addressed for global acceptance of distributed computing at large
- Probabilistic verification $\equiv$ good trade-off result quality/overhead
- Efficient even with only 1 verifier
- HPC confidentiality and remote trust needs hardware support
- SAFESCALE architecture embraces different amounts of secure hardware
  - Pure software execution with verification on her own well controlled machines
  - Pure software execution with verification on some (remote) hardware secured machines
  - Software execution on hardware secured (remote) machines, no need for verification
- KAAPI C++ framework to ease task parallelism
- PIPS-based tool to generate KAAPI code for legacy applications

Thanks for your attention...

Questions?

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 controller calls (ideally $o(n)$)
  - $\hookrightarrow$ **can be done in parallel on R**!

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

$\hookrightarrow$ the application **should** tolerate a limited number of faults [cf. chap.7]

$\hookrightarrow$ no assumption on attackers behaviour

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 work-stealing**
  - $\hookrightarrow$ 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:

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

EMCT algorithm

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^≤(T) on R to detect initiators
forall T_j ∈ G^≤(T) / T_j as not yet been checked do
    ô(T_j, E) ← ReexecuteOnVerifier(T_j, i(T_j, E));
    if o(T_j, E) ≠ ô(T_j, E) then
        return FALSIFIED;
end
return CORRECT;
```

EMCT algorithm (2)

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 $W_\infty^C = \Omega(W_\infty)$
- **Yet** (Trees/F-J graphs): $W_1^C = \mathcal{O}(hW_\infty)$ where h is the height

$EMCT(E)$ 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 $G^{\leq}(T)$

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		$\mathcal{O}(W_\infty)$	$\mathcal{O}(W_\infty)$	$\mathcal{O}(W_\infty)$	$\mathcal{O}(W_\infty)$