Result Certification

- What does one do when applications get large...?
  - The results of a large computation is returned:
    - Is that result correct?
    - Are there computational errors?
    - Has the result been altered by partial manipulation?
    - Has there been a massive attack?
    - ...

Result Certification

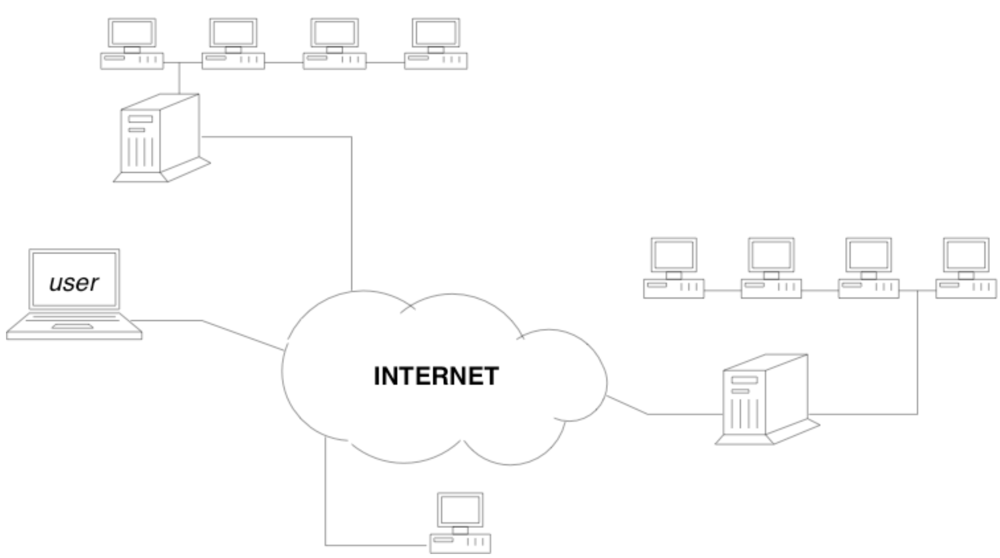
- How do you know whether the results of a large computation have not been corrupted?
  - This sequence is based on
    - Krings Axel W., Jean-Louis Roch, and Samir Jafar, "Certification of Large Distributed Computations with Task Dependencies in Hostile Environments", IEEE Electro/Information Technology Conference , (EIT 2005), May 22-25, Lincoln, Nebraska, 2005
    - Krings Axel, Jean-Louis Roch, Samir Jafar and Sebastien Varrette, "A Probabilistic Approach for Task and Result Certification of Large-scale Distributed Applications in Hostile Environments", Proc. European Grid Conference (EGC2005), in LNCS 3470, Springer Verlag, February 14-16, 2005, Amsterdam, Netherlands.
    - Sarmenta,Luis F.G., Sabotage-ToleranceMechanisms for Volunteer Computing Systems, Future Generation Computer Systems, No. 4, Vol. 18, 2002.

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

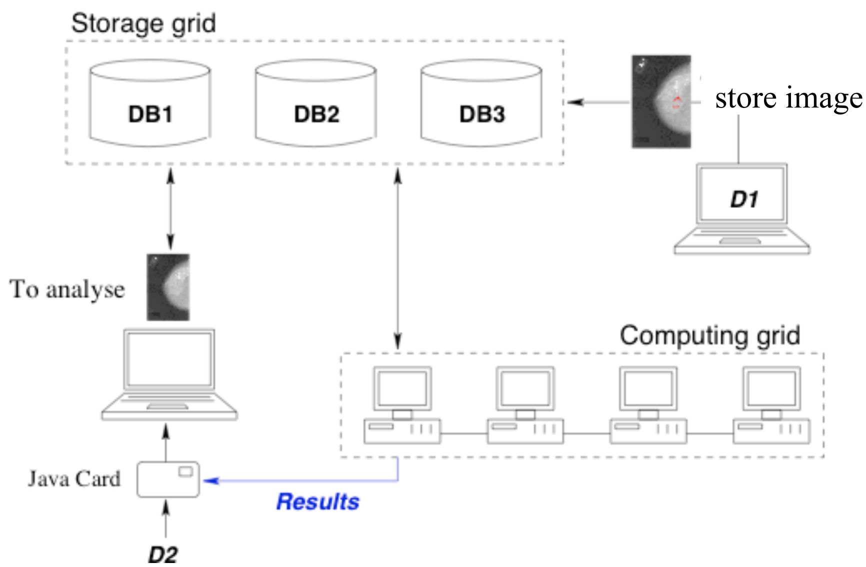
Global Computing Architecture

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



Typical Application

- Computation intensive parallel application
  - e.g. Medical (mammography comparison)



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.

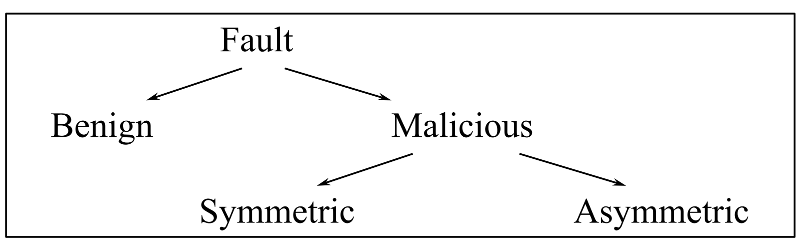
What risks are we subjecting our applications to?

Nodes will fail or be compromised!

- Two important questions:
  - How does one deal with the problem of node failure?
    - Fault-tolerance of "few" failures is built into application
  - Where is the threshold of failures an application can tolerate?
    - Does one know the number of failed nodes or wrong results?

Fault Models: Déjà vu

- Large computations subject to the same spectrum of faults:



- Fault-Behavior and Assumptions
  - Independence of faults
  - Common mode faults -> towards arbitrary faults!
- Fault Sources
  - Trojan, virus, DOS, DDOS, 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
- Focus: massive attacks
  - virus, trojan, DoS, etc.

How does the application survive?

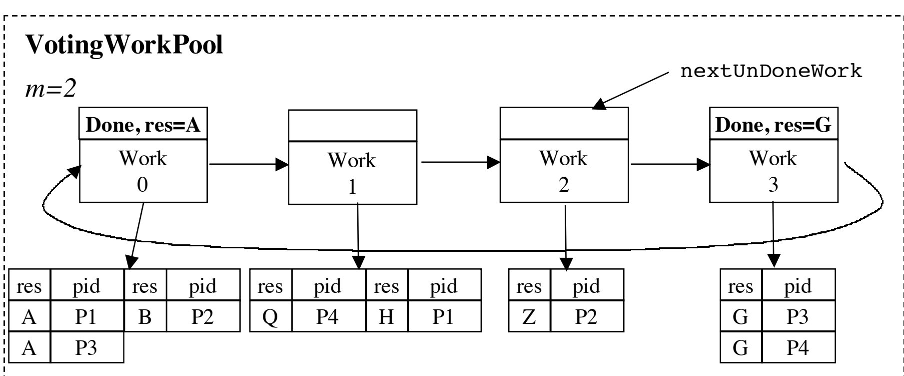
- Key is **Fault Threshold**
- Two main aspects
  1. Application has to be designed to tolerate a certain number of faults
     - implications of infrastructure size on reliability
       - worst case series RBD
     - use fault-tolerance algorithms
       - e.g. fault-tolerant scheduling
  2. One has to detect when fault threshold is surpassed.

Certification Against Attacks

- What is “Certification” in this context?
  - Mainly addressed for independent tasks
- Current approaches
  - Voting
  - Spot-checking
  - Blacklisting
  - Credibility-based fault-tolerance
  - Partial execution on reliable resources (partitioning)
  - Re-execution on reliable resources
- Certification of Computation

Majority Voting

- Compute each piece of work several times
- Decide which result to accept via voting
  - example: modified *eager scheduling work pool*
    - » m=2, 2-first voting scheme
    - » expected redundancy:$m/(1\text{-}f)$, where f is fault fraction



source: Sarmenta2002

Spot-Checking

- Master randomly gives worker a spotter work
  - result is already known
  - if worker is caught with wrong result:
    - » master backtracks through all that worker's results and invalidates them
    - » master may also blacklist the exposed worker from future work
- Has much lower redundancy than voting
  - Redundancy level is: $1/(1-q)$
  - q is the Bernoulli probability of being checked
- Useful if f is large, or maximum acceptable error rate is not too small

Spot-Checking with Blacklisting

- Caught saboteurs are blacklisted
  - not allowed to return to the worker pool
  - assume saboteur receives n work objects (including spotters)
  - then *average final error rate* is

$$\varepsilon_{\text{scbl}}(q,n,f,s) = \frac{sf(1-qs)^n}{(1-f)+f(1-qs)^n}$$

  - s is sabotage rate of a saboteur
  - f is the fraction of the original population that were saboteurs
  - $(1 - qs)^n$ is the probability of a saboteur surviving though n turns
  - denominator represents fraction of original worker population that survive to the end of the batch
  - see Samenta 2002

Credibility-based Fault-Tolerance

- Could combine *voting* and *spot-checking*
  - achieved error rates are orders-of-magnitude smaller
- More general: credibility-based fault-tolerance
  - compute *credibility* of each tentative result as conditional probability that the result is correct
    - based on voting
    - spot-checking
    - other factors, e.g., some workers may be more trustworthy

Partial re-executions

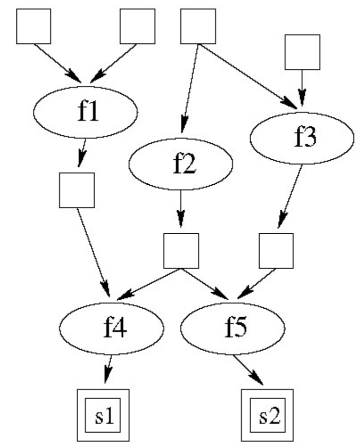
- What is a *reliable* resource?
- Use partitioning
  - execute part of the work on reliable resource
  - execute other parts on normal workers

Execution Model: 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

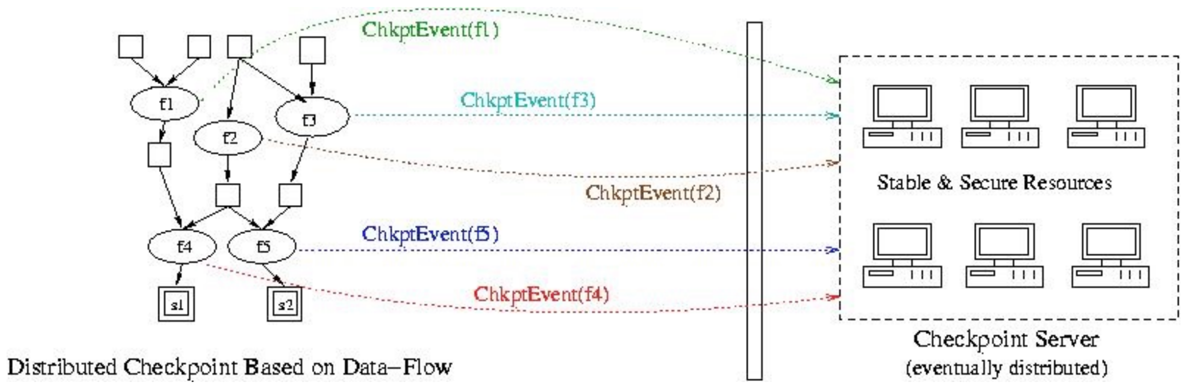
  inputs and outputs

General Execution Environment

- Checkpoint Server: Interface between two environments

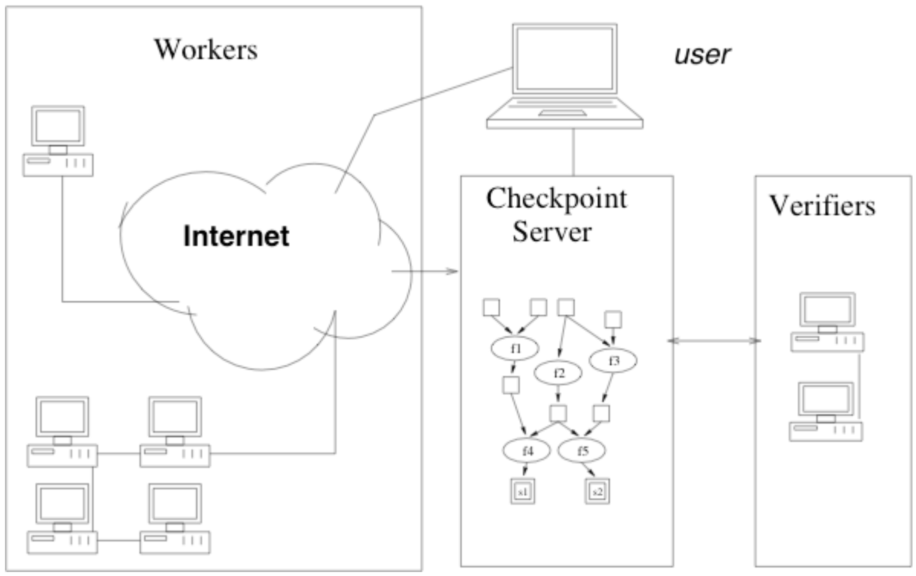
Unreliable Application Execution Environment

Reliable Resources (Verifier)



Global Computing Platform (GCP)

- GCP includes workers, checkpoint server and verifiers



Definitions

- Executions in unreliable environment

E execution of workload represented by *G*

i(T,E) input to *T* in execution *E*

o(T,E) output of *T* in execution *E*

- Executions in reliable environment: Verifier

Ê execution of workload *G* on Verifier

î(T,Ê) input to *T* in execution *Ê*

ô(T,Ê) output of *T* in execution *Ê*

ô(T,E) output of *T* with input from *E* executing on verifier

Note: notations *ô(T,Ê)* and *ô(T,E)* differ!

- If $E = \hat{E}$ then *E* is said to be "correct"
  otherwise *E* is said to have "failed"

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.

Probabilistic Certification

- What does the certification really mean?
  - what is the real interpretation of $E = \hat{E}$
  - connection between $E = \hat{E}$ and massive attack
  - use $E = \hat{E}$ as a "tool" to determine if a massive attack has occurred
- Monte Carlo certification against massive attacks
  - number of tasks actually failed/attacked n_F
  - consider two scenarios
    - » $n_F = 0$
    - » n_F is large => massive attack
- Attack Ratio q $\quad n_q = \lceil nq \rceil \leq n_F$

Monte Carlo Test

- Algorithm MCT
  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
  - Return CORRECT

- Assume all tasks in G are independent
  1. we always have $i(T,E) = \hat{\imath}(T,\hat{E})$

Certification of Independent Tasks

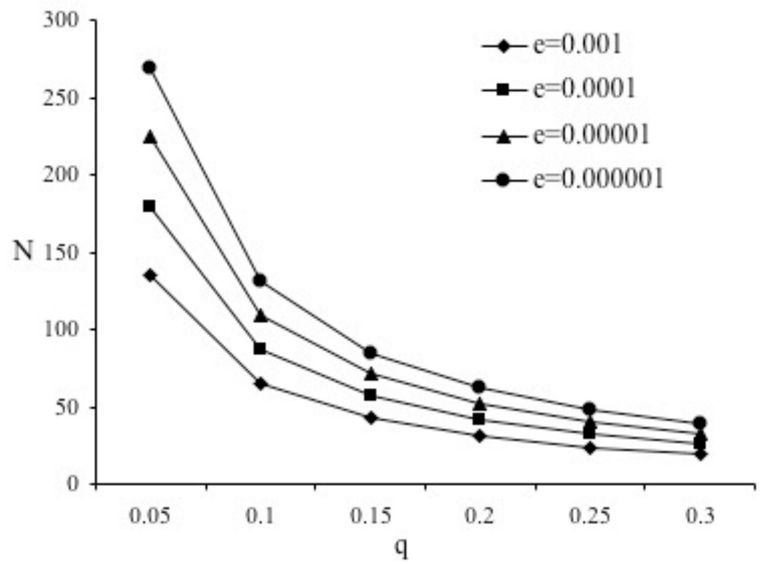
- Main Result
  - *Let E be an execution with n independent tasks and assume that E is either correct or massively attacked with ratio q. For a given ε, the number of independent executions of algorithm MCT necessary to achieve a certification of E with probability of error less than or equal to ε is*

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

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

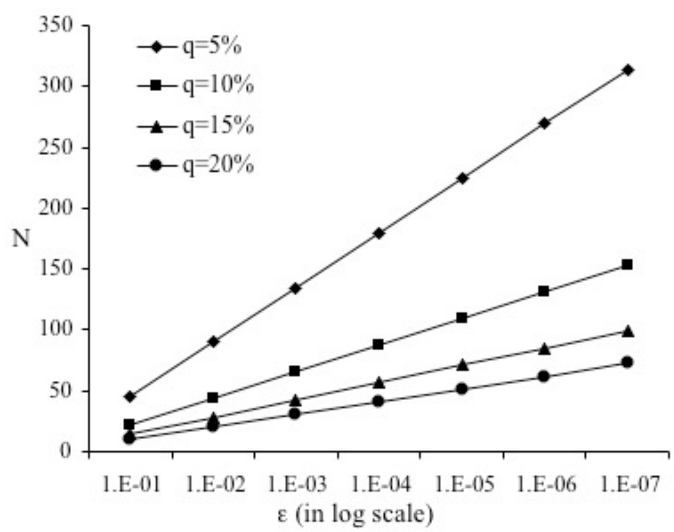
Certification of Independent Tasks

- Relationship between attack ratio and N



Certification of Independent Tasks

- Relationship between certification error and N



Certification with task dependencies

- **What changes when one considers task dependance?**

Certification and Task Dependencies

- What does a re-execution really tell us w.r.t. the result?
  - One can only talk about outputs of tasks, not tasks!
  - If $o(T,E) \neq \hat{o}(T,E)$ we know that an error has occurred
  - If $o(T,E) = \hat{o}(T,E)$ we cannot say much at all!
    - » for independent tasks this indicated a good task/result
    - » what do we know about the inputs?
      - in the presence of error propagation -- not much!
    - » if the verifier uses $\hat{\imath}(T,\hat{E})$ then $o(T,E) = \hat{o}(T,\hat{E})$ indicates a good result

      but we don't have $\hat{E}$, (would require total re-execution on verifier)

Certification and Task Dependencies

- The concept of "Initiator"
  - $o(T,E) = \hat{o}(T,E)$ is only useful if we know that the inputs are correct
    - this implies that T has no forged predecessors
  - Definition:

    An ***initiator*** is a falsifying task that has no falsifying predecessors
  - Worst case assumption is very conservative
    - one still might detect a falsified non-initiator
    - but there is not guarantee

Certification and Task Dependencies

- Certification is now based on initiators
- *Lemma 2*
  - *The probability that MCT return FAILED is at least n_I/n and the probability it returns CORRECT is $\leq 1 - n_I/n$*
- *Lemma 3*
  - *Let E be an execution of tasks with dependencies and assume that E is either correct or massively attacked with ration q. For a given ε, the number of independent executions of algorithm MCT necessary to achieve a certification of E with probability of error less than or equal to ε is*

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

Certification and Task Dependencies

$G^{\leq}(T)$	predecessor graph of T
V	a set of tasks in G
$G^{\leq}(V)$	predecessor graph of all tasks in V
$k \leq n_F$	be the number of falsified tasks assumed
$I(F)$	set of all initiators

- Minimum Number of Initiators

$$\gamma_V(k) = \min |G^{\leq}(V) \cap I(F)|$$

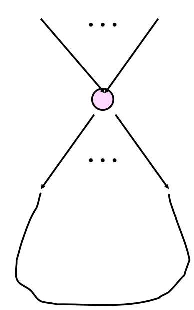
- Minimal Initiator Ratio

$$\Gamma_V(k) = \frac{\gamma_V(k)}{|G^{\leq}(V)|}$$

Certification and Task Dependencies

- The impact of graph G
  - Knowing the graph, an attacker may attempt to minimize the visibility of even a massive attack with ration q.
  - What is the number of initiators one might have to expect in a graph?
    - Given height h (the length of the critical path) and maximum out- degree d of a graph G, the minimum number of initiators is

$$\gamma_G(n_F) = \left\lceil \frac{n_F}{\left(\frac{1-d^h}{1-d}\right)} \right\rceil$$



Extended Monte Carlo Test

- 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

1. Behavior
   1. disadvantage: the entire predecessor graph needs to be re-executed
   2. however: the cost depends on the graph
      1. luckily our application graphs are mainly trees

Analysis of EMCT

- Probability of error for single execution:
  - worst case
    - forged tasks are distributed to minimize the number of T whose $G^{\leq}(T)$ contain falsified tasks
    - this is the case when the attack is biased towards leaf nodes
  - error probability $e_E \leq 1 - q$

Analysis of EMCT

- What is the cost (number of verifications) of a single invocation:
  - exact number of verifications is known only at run-time
    - depends on which T is selected

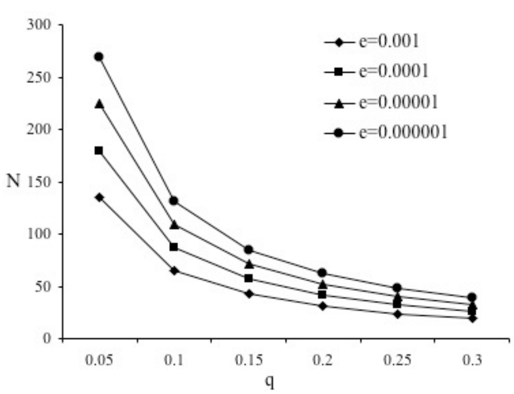
$$C = |G^{\leq}(T)|$$

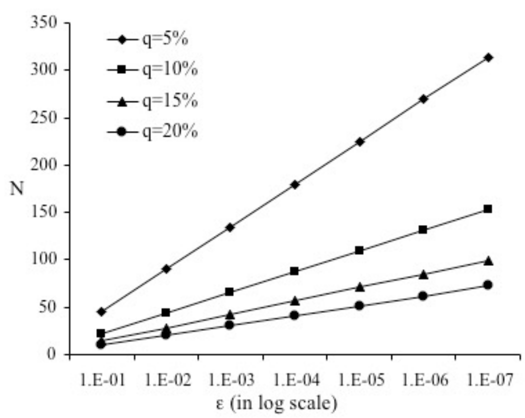
  - expected number of verifications:
    - average number of tasks in a predecessor graph, over all T_i in G.

$$C = \frac{\sum_{T_i \in G} |G^{\leq}(T_i)|}{n}$$

Analysis of EMCT

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





Results for MCT and EMTC

- Considered
  - General graphs
  - Out-trees (application domain based on out/in-trees)

Algorithm	MCT	$EMCT$
Number of effective initiators	$\left\lceil \frac{n_q}{\left(\frac{1-d^h}{1-d}\right)} \right\rceil$	n_q
Probability of error	$1 - \frac{\left\lceil \frac{n_q}{\left(\frac{1-d^h}{1-d}\right)} \right\rceil}{n}$	$1 - q$
Verification cost: general G	1	$O(n)$
Verification cost: G is out-tree	1	$h - log_d(n_v)$
Ave. # effective initiators, G is out-tree	$\left\lceil \frac{n_q}{\left(\frac{1-(h+2)d^{h+1}+(h+1)d^{h+2}}{(1-d)(1-d^{h+1})}\right)} \right\rceil$	n_q

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

Relationship between quantities

- Given a subset V of tasks in G.

  What are the relationships between
  $\gamma_V(k)$, $\gamma_G(k)$ and n_I with respect to $k = n_q$ or $k = n_F$?

  By definition
  $q \leq n_F / n$ and thus $n_q \leq n_F$
  also
  $n_I \leq n_F$

Relationship between quantities

- With respect to n_F we always have
  $$\gamma_V(n_F) \leq \gamma_G(n_F) \leq n_I \leq n_F$$
  - But where does n_q fit into this inequality?
  - The only certain relationship is $n_q \leq n_F$
- With respect to n_q we always have
  $$\gamma_V(n_q) \leq \gamma_G(n_q) \leq n_q \leq n_F$$
  - But where does n_I fit into this inequality?
  - The only certain relationship is $\gamma_G(n_q) \leq n_I \leq n_F$

Relationship between quantities

- With respect to $n_q \le n_F$ we can compare directly

$$\gamma_V(n_q) \le \gamma_V(n_F)$$
$$\gamma_G(n_q) \le \gamma_G(n_F)$$

Thus

$$\Gamma_V(n_q) \le \Gamma_V(n_F)$$
$$\Gamma_G(n_q) \le \Gamma_G(n_F)$$

Verifying with fractions of $G^{\le}(T)$

- We will now modify algorithm EMCT so that only a fraction of tasks in the predecessors are verified.

Verifying with fractions of $G^{\leq}(T)$

- Algorithm EMCTα(*E*)

1. Uniformly choose one task T in G.
2. Uniformly select $n_\alpha = \lceil \alpha |G^{\leq}(T)| \rceil$ tasks in $G^{\leq}(T)$ and let this set be denoted by A. If for any $T_j \in A$, that has not been verified yet, re-execution on a verifier results in $\hat{o}(T_j, E) \neq o(T_j, E)$ then return FAILED.
3. Return CORRECT.

Verifying with fractions of $G^{\leq}(T)$

- For Algorithm EMCTα(*E*)

Lemma 1 *Let T be a task randomly chosen by $EMCT_\alpha(E)$. Then the probability of error, e_α, when $EMCT_\alpha(E)$ returns CORRECT is given by*

$$e_\alpha \leq \begin{cases} (1 - q\alpha\Gamma_T(n_q)) & \text{for} \quad 0 < \alpha \leq 1 - \Gamma_T(n_q) \\ (1-q) & \text{otherwise.} \end{cases}$$

Verifying with fractions of $G^{\leq}(T)$

- For Algorithm EMCTα(*E*)

Theorem 1 *Let E be an execution with dependencies that is either correct or massively attacked with ratio q. Given ϵ and $0 < \alpha \leq 1$, N independent invocations of Algorithm $EMCT_\alpha(E)$ provide a certification with error probability*

$$\epsilon \leq \begin{cases} (1 - q\alpha\Gamma_G(n_q))^N & for\ 0 < \alpha \leq 1 - \Gamma_T(n_q) \\ (1-q)^N & otherwise. \end{cases}$$

Verifying fixed numbers of tasks

- We will now modify algorithm EMCT so that only a fixed number of tasks in the predecessors are verified.
  - We limit our investigations to unity, i.e. one task is verified.

Verifying fixed numbers of tasks

- Algorithm EMCT[1](E)

1. Uniformly choose one task T in G.
2. Uniformly select a single T_j in $G^{\leq}(T)$. If re-execution of T_j on a verifier results in $\hat{o}(T_j, E) \neq o(T_j, E)$ then return FAILED.
3. Return CORRECT.

Verifying fixed numbers of tasks

- For Algorithm EMCT[1](E)

Lemma 2 *Let T be a task randomly chosen by $EMCT^1(E)$ and let $V = G^{\leq}(T)$. Then the probability of error, e_1, when $EMCT^1(E)$ returns CORRECT is given by*

$$e_1 \leq 1 - \frac{n_F}{n}\Gamma_T(n_F) \leq 1 - q\Gamma_T(n_q)$$

Verifying fixed numbers of tasks

- For Algorithm EMCT[1](E)

Theorem 2 *Let E be an execution with dependencies that is either correct or massively attacked with ratio q. Given ϵ then N independent invocations of Algorithm $EMCT^1(E)$ provide a certification with error probability*

$$\epsilon \le (1 - q\Gamma_G(n_q))^N.$$

The cost of certification

- A balance between N and C
- Monte Carlo certification for a given ε:
  1. a priori convergence
     - determine up front how many times one has to verify
     - one does not know which tasks are selected
  2. run-time convergence
     - run until certain ε is achieved
     - take advantage of knowledge about task selected
  3. for general graphs
  4. for special graphs (e.g. out-trees)

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	$\frac{\log \epsilon}{\log\left(1 - \frac{\left\lceil \frac{n_q}{\left(\frac{1-d^h}{1-d}\right)} \right\rceil}{n}\right)}$	$\frac{\log \epsilon}{\log(1-q)}$	$\frac{\log \epsilon}{\log(1-q\alpha\Gamma_G(n_q))}$ 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

Results for pathological cases

- Probability of error induced by one invocation
  - derived for each algorithm

	$MCT(E)$ [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	$\frac{\log \epsilon}{\log\left(1 - \frac{\left\lceil \frac{n_q}{\left(\frac{1-d^h}{1-d}\right)} \right\rceil}{n}\right)}$	$\frac{\log \epsilon}{\log(1-q)}$	$\frac{\log \epsilon}{\log(1-q\alpha\Gamma_G(n_q))}$ 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

Results for pathological cases

- A priori convergence (N is determined a priori)
  - cannot take advantage of run-time knowledge
  - has to use $\Gamma_G(n_q)$ rather than $\Gamma_T(n_q)$
  - q_e is the effective attack ratio

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

	$MCT(E)$ [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	$\frac{\log \epsilon}{\log\left(1 - \frac{\left\lceil \frac{n_q}{\left(\frac{1-d^h}{1-d}\right)} \right\rceil}{n}\right)}$	$\frac{\log \epsilon}{\log(1-q)}$	$\frac{\log \epsilon}{\log(1-q\alpha\Gamma_G(n_q))}$ 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

Results for pathological cases

- Run-time convergence (N is determined at run-time)
  - takes advantage of run-time knowledge
  - initial verification $\varepsilon_e = 1 - q_e$
  - each verification $\varepsilon_e = \varepsilon_e (1 - q_e)$
  - untile $\varepsilon_e \leq \varepsilon$

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

	$MCT(E)$ [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	$\frac{\log \epsilon}{\log\left(1 - \frac{\left\lceil \frac{n_q}{\left(\frac{1-d^h}{1-d}\right)} \right\rceil}{n}\right)}$	$\frac{\log \epsilon}{\log(1-q)}$	$\frac{\log \epsilon}{\log(1-q\alpha\Gamma_G(n_q))}$ 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

Results for pathological cases

- Verification cost
  - per invocation of the algorithm
  - special case: out-tree

	$MCT(E)$ [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	$\frac{\log \epsilon}{\log\left(1 - \frac{\left\lceil \frac{n_q}{\left(\frac{1-d^h}{1-d}\right)} \right\rceil}{n}\right)}$	$\frac{\log \epsilon}{\log(1-q)}$	$\frac{\log \epsilon}{\log(1-q\alpha\Gamma_G(n_q))}$ 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	$\lvert G^{\le}(T)\rvert$	$\lceil \alpha \lvert G^{\le}(T)\rvert \rceil$	1
Max. cost (out-tree)	1	h	αh	1

Conclusions

- Certification of large distributed applications
  - hostile environments with no assumptions on fault model
- Considered task dependencies
  - tasks or data may be manipulated
  - allows for error propagation (much more difficult than independent case)
  - very difficult to speculate on the behavior of a falsified task
- Several probabilistic certification algorithms were introduced
  - based on re-execution on verifier (reliable resource)
  - inputs available from dataflow checkpoints
- Certification:
  - very low probability of error can be achieved
  - number of tasks to verify is relatively small, depending on graph
  - relationship between attack rate and probability of error