Adaptive redundant residue system for cloud computing: a processor-oblivious and fault-oblivious technology

Jean-Louis ROCH
*INRIA **MOAIS** team, LIG lab.*
Grenoble Université, France

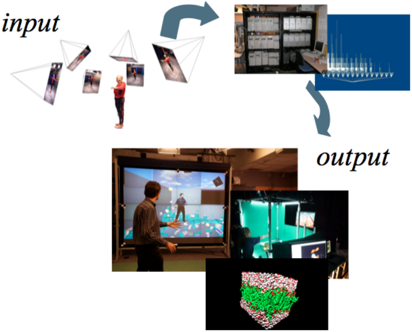
Joint work with
C. Pernet [1], T. Roche [1, 4], S. Varrette [3, 2], S. Jafar [2], A. Krings [2], B. Cunche [4], M. Khonji[1], T. Stalinski[1]

Computer Science Dept, University of Western Ontario
Monday January 24, 2011

Plan

Context: High Performance Interactive Computation

INRIA - LIG Moais team



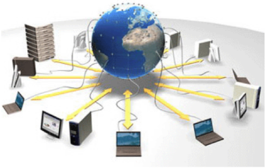
- Performance: multi-criteria trade-off
  $\hookrightarrow$ computation latency and computation bandwidth
- Application domains: interactive simulations (latency), computer algebra (bandwidth), ...

HPC platforms

From low computation latency to high computation bandwidth

- **Parallel chips & multi/many-core architectures**: multicore cpu, GP-GPU, FPGA, MPSoCs
- **Servers**: Multi-processor with ”sharedi” memory (CPUs + GPUs+...)
- **Clusters**: 72% of top 500 machines, Heterogeneous (CPUs + GPUs + ...)
  E.g. Tianhe-1A, ranked 1 in TOP500 nov 2010: 4.7 PFlops
- **Global computing platforms**: grids, P2P, **clouds**
  E.g. BOINC : in April 2010= 5.1 PFlops

Cloud / Global computing platforms



Applications

- Data sharing (1980's), Data storage, Computation (1990's)

"Unbound" computation bandwidth

- *Volunteer Computing*: steal idle cycles through the Internet
  - Folding@Home – 5 PFLOPS, as of March 17, 2009
  - MilkyWay@Home - 1.6 PFLOPS, as of April 2010
    Thanks to GPUs !

Grid or cloud ?

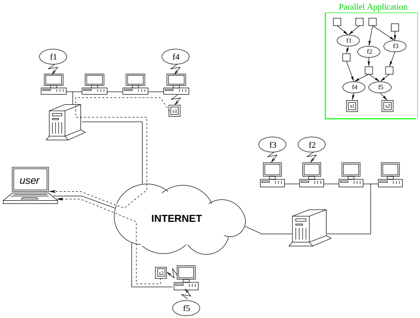
Beyond computation bandwidth, two important criteria:

- **granularity**: how small is the smallest bit of computational resource that you can buy;
- **speed of scaling**: how long it takes to increase the size of available resources.

Global computing architecture and trust

Open platforms are subject to attacks:

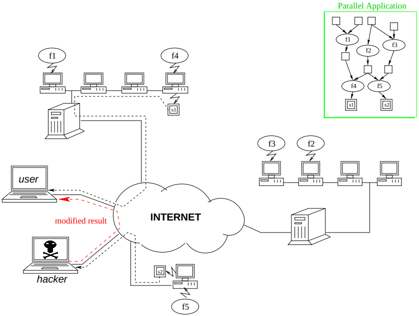
- machine badly configured, over clocking,
- malicious programs,
- client patched and redistributed: possibly large scale



Global computing architecture and trust

Open platforms are subject to attacks:

- machine badly configured, over clocking,
- malicious programs,
- client patched and redistributed: possibly large scale



Global computing platforms: drawbacks

Peers volatility

Peers can join/leave at any time.

Faults, crashes

As the number of nodes increases, the number of faults, process crashes increases as well.

Trust in Peers

Malicious Peers (intentionally or infected by malwares).

- Random Crashes
- Random Forgeries
- Byzantine behaviour (e.g., Peers collusion)

Related work: trusting the Cloud

Volatility

Replication, (partial) Re-execution, Checkpoint/restart

Trust

- Challenges / blacklisting [Sramenta&ak 01]
- Replication of tasks:
  - BOINC: credit evaluation
  - Byzantine agreement: n processes for $n/3$ faulty (one-third faulty replicas [Lamport82s])
- Check / Verification using postconditions (on the output) [Blum97]

Related work: trusting the Cloud

Volatility

Replication, (partial) Re-execution, Checkpoint/restart

Trust

- Challenges / blacklisting [Sramenta&ak 01]
  ⇒expensive
- Replication of tasks:
  - BOINC: credit evaluation
    ⇒Attacks can be adjusted
  - Byzantine agreement: n processes for $n/3$ faulty (one-third faulty replicas [Lamport82s])
    ⇒often expensive
- Check / Verification using postconditions (on the output) [Blum97]
  ⇒not always possible

Related work: trusting the Cloud

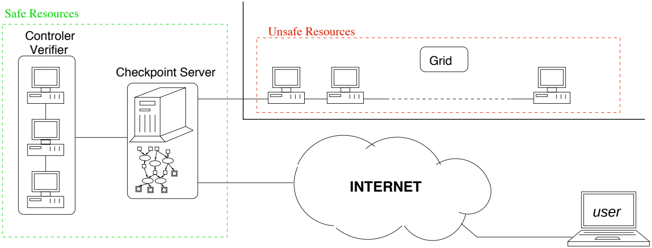
- Without trust assumptions, basic properties (eg integrity, atomicity, weak consistency, ...) cannot be guaranteed
- But, by maintaining a small amount of trusted memory and trusted computation units,
  well-known cryptographic methods reduce the need for trust in the storage [Cachin&al. ACM SIGACT 2009]
  - integrity by storing small hash (hash tree for huge data)
  - authentication of data
  - proofs of retrievability (POR) and of data possession (PDP)

Considered cloud computing platform

$\Longrightarrow$ Two disjoint set of resources:

- U: The cloud, unreliable
- R: Resources blindly trusted by the user (reliable), but with limited computation bandwidth

interconnected through a stable memory



Plan

How to take benefit of the Cloud for HPiC?

Performance and correction

- **Performance:** massive parallelism (*work* $\gg$ *depth*)
  $\hookrightarrow$ workstealing, adaptive scheduling [Leiserson&al. 2007]
- **Correction of the results:** proof
  $\hookrightarrow$ verifications on R (e.g. randomized checking)
- **Fault tolerance:** to support resilience and/or errors
  $\hookrightarrow$ ABFT: Algorithm Based Fault Tolerance

Execution time model on the cloud

Resource type	average computation bandwidth per proc	Total computation bandwidth	Usage
Cloud U	Π_U	Π_U^{tot}	computation
Client R	Π_R	Π_R^{tot}	certification

bandwidth: number of unit operations per second

Bound on the time required for computation+certification

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

$$\text{Execution time} \leq \frac{W_c}{\Pi_U^{tot}} + \mathcal{O}\left(\frac{D_c}{\Pi_U}\right) + \frac{W_r}{\Pi_R^{tot}} + \mathcal{O}\left(\frac{D_r}{\Pi_R}\right).$$

Notations and target context:

- W_c (resp. D_c) = total work (resp. depth) executed on the cloud U;
- W_r (resp. D_r) = total work (resp. depth) executed on the client R;
- Target context: $W_c = O(W_1^{1+\epsilon})$; $D_c \ll W_c$; $W_r, D_r \ll D_c$.
- Correction: may be computed either on R, or on U, or on both.

Plan

ABFT: Algorithmic Based Fault Tolerance

Idea: incorporate redundancy in the algorithm

[Huang&Abraham 98] [Saha 2006] [Dongarra & al. 2006]

$\Rightarrow$use properties specific to the problem

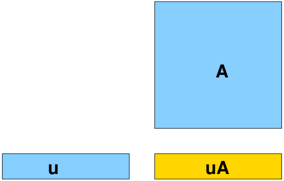
ABFT: Algorithmic Based Fault Tolerance

Idea: incorporate redundancy in the algorithm

[Huang&Abraham 98] [Saha 2006] [Dongarra & al. 2006]

⇒use properties specific to the problem

Example: Matrix-vector product



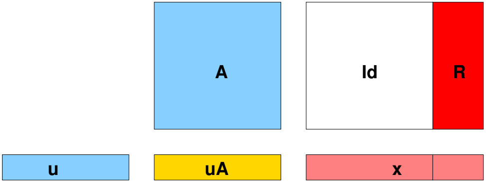
ABFT: Algorithmic Based Fault Tolerance

Idea: incorporate redundancy in the algorithm

[Huang&Abraham 98] [Saha 2006] [Dongarra & al. 2006]

$\Rightarrow$ use properties specific to the problem

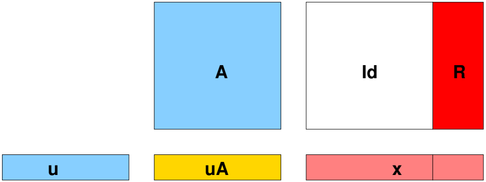
Example: Matrix-vector product



ABFT: Algorithmic Based Fault Tolerance

Idea: incorporate redundancy in the algorithm
[Huang&Abraham 98] [Saha 2006] [Dongarra & al. 2006]
$\Rightarrow$use properties specific to the problem

Example: Matrix-vector product



- pre-compute the product $B = A \times \begin{bmatrix} I & R \end{bmatrix}$
- compute $x = uB$ in parallel
- decode/correct x

Summary

Typical considered global computation

- Interactive computation between R (reliable) and the cloud U (unreliable);
- On U: Computations loosely coupled
  and fault tolerant
- On R: submission of the computation
  correction (if not too much errors)
  verification (if not too much errors)

$\hookrightarrow$ homorphic residue system

Plan

Homomorphic residue system

- One-to-one mapping Φ from E to $E_1 \times \ldots \times E_n$ that preserves the algebraic structure
- $\Phi : x \mapsto x_1, \ldots x_n$: projection
  $\Phi^{-1}(x_1, \ldots, x_n)$: lifting
- $\hookrightarrow$ For a fixed computation P (straight-line program):
  For input x: $\Phi(P_x) = \Phi(P)_{x_1}, \ldots, \Phi(P)_{x_n}$.
- Classical examples: residue number/polynomial systems:
  - Polynomials: $\Phi(Q) = (Q(a_1), \ldots, Q_(a_n))$ [evaluation/interpolation]
  - **Integers**: $\Phi(x) = (x \mod a_1, \ldots, x \mod a_n)$
    $a_1, \ldots, a_n$: residue number system (relatively prime)

Integers: Chinese remainder algorithm

$$\mathbb{Z}/(n_1 \dots n_k)\mathbb{Z} \equiv \mathbb{Z}/n_1\mathbb{Z} \times \cdots \times \mathbb{Z}/n_k\mathbb{Z}$$

Computation of $y = f(x)$ over $\mathbb{Z}$

begin
- Compute a bound β on $\max(|f|)$;
- Pick $n_1, \dots n_k$, pairwise prime, s.t. $n_1 \dots n_k > \beta$;
- **for** $i = 1 \dots k$ **do**
  - Compute $y_i = f(x \mod n_i) \mod n_i$
- Compute $y = \mathrm{CRT}(y_1, \dots, y_k)$

end

$$\begin{array}{rrcl} \mathrm{CRT}: & \mathbb{Z}/n_1\mathbb{Z} \times \cdots \times \mathbb{Z}/n_k\mathbb{Z} & \rightarrow & \mathbb{Z}/(n_1 \dots n_k)\mathbb{Z} \\ & (x_1, \dots, x_k) & \mapsto & \sum_{i=1}^{k} x_i \Pi_i Y_i \mod \Pi \end{array}$$

$$\text{where} \left\{ \begin{array}{rcl} \Pi & = & \prod_{i=1}^{k} n_i \\ \Pi_i & = & \Pi / n_i \\ Y_i & = & \Pi_i^{-1} \mod n_i \end{array} \right.$$

Integers: Chinese remainder algorithm

$$\mathbb{Z}/(n_1 \dots n_k)\mathbb{Z} \equiv \mathbb{Z}/n_1\mathbb{Z} \times \cdots \times \mathbb{Z}/n_k\mathbb{Z}$$

Computation of $y = f(x)$ over $\mathbb{Z}$

begin
  Compute a bound β on $\max(|f|)$;
  Pick $n_1, \dots n_k$, pairwise prime, s.t. $n_1 \dots n_k > \beta$;
  for $i = 1 \dots k$ **do**
    Compute $y_i = f(x \mod n_i) \mod n_i$;   /* Evaluation */
  Compute $y = \mathrm{CRT}(y_1, \dots, y_k)$;   /* Interpolation */
end

$$\begin{array}{rrcl} \mathrm{CRT}: & \mathbb{Z}/n_1\mathbb{Z} \times \cdots \times \mathbb{Z}/n_k\mathbb{Z} & \rightarrow & \mathbb{Z}/(n_1 \dots n_k)\mathbb{Z} \\ & (x_1, \dots, x_k) & \mapsto & \sum_{i=1}^{k} x_i \Pi_i Y_i \mod \Pi \end{array}$$

$$\text{where} \begin{cases} \Pi & = & \prod_{i=1}^{k} n_i \\ \Pi_i & = & \Pi / n_i \\ Y_i & = & \Pi_i^{-1} \mod n_i \end{cases}$$

Chinese remaindering and evaluation/interpolation

Evaluate P in a $\leftrightarrow$ Reduce P modulo $X - a$

Chinese remaindering and evaluation/interpolation

Evaluate P in a $\leftrightarrow$ Reduce P modulo $X - a$

Polynomials	
Evaluation: $P \mod X - a$ Evaluate P in a	
Interpolation: $P = \sum_{i=1}^{k} \frac{\prod_{j \neq i}(X - a_j)}{\prod_{j \neq i}(a_i - a_j)}$	

Chinese remaindering and evaluation/interpolation

Evaluate P in a $\leftrightarrow$ Reduce P modulo $X - a$

Polynomials	**Integers**
Evaluation: $P \mod X - a$ Evaluate P in a	$N \mod m$ "Evaluate' N in m
Interpolation: $P = \sum_{i=1}^{k} \frac{\prod_{j \neq i}(X - a_j)}{\prod_{j \neq i}(a_i - a_j)}$	$N = \sum_{i=1}^{k} a_i \prod_{j \neq i} m_j (\prod_{j \neq i} m_j)^{-1 [m_i]}$

Chinese remaindering and evaluation/interpolation

Evaluate P in a $\leftrightarrow$ Reduce P modulo $X - a$

Polynomials	**Integers**
Evaluation: $P \mod X - a$ Evaluate P in a	$N \mod m$ "Evaluate' N in m
Interpolation: $P = \sum_{i=1}^{k} \frac{\prod_{j \neq i}(X - a_j)}{\prod_{j \neq i}(a_i - a_j)}$	$N = \sum_{i=1}^{k} a_i \prod_{j \neq i} m_j (\prod_{j \neq i} m_j)^{-1 [m_i]}$

Analogy: complexities over $\mathbb{Z} \leftrightarrow$ over $K[X]$

For a program with $T_{\text{algebr.}}$ algebraic operations:

- size of coefficients
- $\mathcal{O}\left(\log \|\text{result}\| \times T_{\text{algebr.}}\right)$

- degree of polynomials
- $\mathcal{O}\left(\deg(\text{result}) \times T_{\text{algebr.}}\right)$

Chinese remaindering and evaluation/interpolation

Evaluate P in a $\leftrightarrow$ Reduce P modulo $X - a$

Polynomials	**Integers**
Evaluation: $P \mod X - a$ Evaluate P in a	$N \mod m$ "Evaluate' N in m
Interpolation: $P = \sum_{i=1}^{k} \frac{\prod_{j\neq i}(X-a_j)}{\prod_{j\neq i}(a_i-a_j)}$	$N = \sum_{i=1}^{k} a_i \prod_{j\neq i} m_j (\prod_{j\neq i} m_j)^{-1 [m_i]}$

Analogy: complexities over $\mathbb{Z} \leftrightarrow$ over $K[X]$

For a program with $T_{\text{algebr.}}$ algebraic operations:

- size of coefficients
- $\mathcal{O}\left(\log \|\text{result}\| \times T_{\text{algebr.}}\right)$
- $\det(n, \|A\|) = \tilde{\mathcal{O}}\left(n \log \|A\| \times n^{\omega}\right)$

- degree of polynomials
- $\mathcal{O}\left(\deg(\text{result}) \times T_{\text{algebr.}}\right)$
- $\det(n, d) = \tilde{\mathcal{O}}\left(nd \times n^{\omega}\right)$

Early termination

Classic Chinese remaindering

- bound β on the result
- Choice of the n_i: such that $n_1 \dots n_k > \beta$

$\Rightarrow$deterministic algorithm

Early termination

Classic Chinese remaindering

- bound β on the result
- Choice of the n_i: such that $n_1 \ldots n_k > \beta$

$\Rightarrow$deterministic algorithm

Early termination

- For each new modulo n_i:
  - reconstruct $y_i = f(x) \mod n_1 \times \cdots \times n_i$
  - If $y_i = y_{i-1}$ $\Rightarrow$terminated

$\Rightarrow$if n_i chosen at random: randomized algorithm (Monte Carlo)

Early termination

Classic Chinese remaindering

- bound β on the result
- Choice of the n_i: such that $n_1 \dots n_k > \beta$

$\Rightarrow$deterministic algorithm

Early termination

- For each new modulo n_i:
  - reconstruct $y_i = f(x) \mod n_1 \times \cdots \times n_i$
  - If $y_i = y_{i-1}$ $\Rightarrow$terminated

$\Rightarrow$if n_i chosen at random: randomized algorithm (Monte Carlo)

Advantage:

- Adaptive number of moduli depending on the output value
- Interesting when
  - pessimistic bound: sparse/structured matrices, ...
  - no bound available

Redundant residues codes

Principle:

- Chinese remaindering based parallelization
- Byzantines faults affecting some modular computations
- Fault tolerant reconstruction
  $\Rightarrow$Algorithm Based Fault Tolerance (ABFT)

Plan

Mandelbaum algorithm over $\mathbb{Z}$

Chinese Remainder Theorem

$$x \in \mathbb{Z} \quad \longleftrightarrow \quad \begin{array}{|c|c|c|c|}\hline x_1 & x_2 & \quad \dots \quad & x_k \\ \hline \end{array}$$

where $p_1 \times \cdots \times p_k > x$ and $x_i = x \mod p_i \, \forall i$

Mandelbaum algorithm over $\mathbb{Z}$

Chinese Remainder Theorem

$x \in \mathbb{Z} \longleftrightarrow$

x_1	x_2	$\dots$	x_k	x_{k+1}	$\dots$	x_n

where $p_1 \times \cdots \times p_n > x$ and $x_i = x \mod p_i \, \forall i$

Mandelbaum algorithm over $\mathbb{Z}$

Chinese Remainder Theorem

$x \in \mathbb{Z} \quad \longleftrightarrow \quad$

x_1	x_2	$\ldots$	x_k	x_{k+1}	$\ldots$	x_n

where $p_1 \times \cdots \times p_n > x$ and $x_i = x \mod p_i \, \forall i$

Definition

(n, k)-code: $C =$

$$\left\{ (x_1, \ldots, x_n) \in \mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_n} \text{ s.t. } \exists! x, \left\{ \begin{array}{lll} x & < & p_1 \ldots p_k \\ x_i & = & x \mod p_i \, \forall i \end{array} \right. \right\}$$

Principle

Property

$$X \in C \textit{ iff } X < \Pi_k.$$

$$\overbrace{\underbrace{\boxed{p_1}\boxed{p_2}\boxed{\quad\dots\quad}\boxed{p_k}}_{\Pi_k = p_1 \times \cdots \times p_k}\boxed{p_{k+1}}\boxed{\dots}\boxed{p_n}}^{\Pi_n = p_1 \times \cdots \times p_n}$$

$$\text{Redundancy} : r = n - k$$

Principle

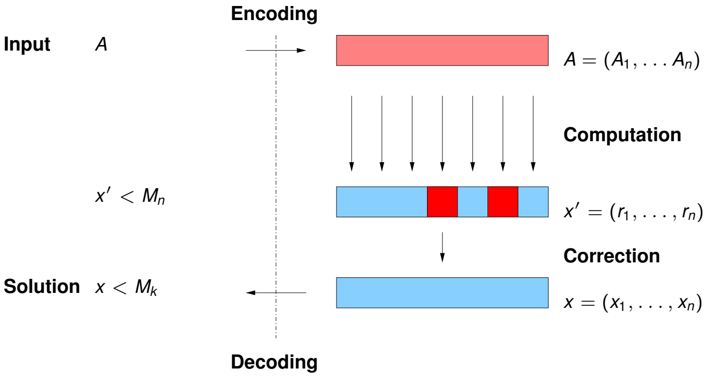
Transmission Channel $\equiv$ Computation

Principle

Noisy Transmission Channel $\equiv$ Unsecured Computation

Principle

Noisy Transmission Channel $\equiv$ Unsecured Computation



Properties of the code

Error model:

- Error: $E = X' - X$
- Error support: $I = \{i \in 1 \ldots n, E \neq 0 \mod p_i\}$
- Impact of the error: $\Pi_F = \prod_{i \in I} p_i$

Properties of the code

Error model:

- Error: $E = X' - X$
- Error support: $I = \{i \in 1 \ldots n, E \neq 0 \mod p_i\}$
- Impact of the error: $\Pi_F = \prod_{i \in I} p_i$

Detects up to r errors:

If $X' = X + E$ with $X \in C, \#I \leq r$,

$$X' > \Pi_k.$$

- Redundancy $r = n - k$, distance: $r + 1$
- $\Rightarrow$can correct up to $\left\lfloor \frac{r}{2} \right\rfloor$ errors in theory
- More complicated in practice...

Correction

- $\forall i \notin I : E \mod p_i = 0$
- E is a multiple of Π_V: $E = Z\Pi_V = Z\prod_{i\notin I}$
- $\gcd(E, \Pi) = \Pi_V$

Mandelbaum 78: rational reconstruction

$$\begin{aligned} X = X' - E &= X' - Z\Pi_V \\ \frac{X}{\Pi} &= \frac{X'}{\Pi} - \frac{Z}{\Pi_F} \end{aligned}$$

$\Rightarrow \left|\frac{X'}{\Pi} - \frac{Z}{\Pi_F}\right| \leq \frac{1}{2\Pi_F^2}$

$\Rightarrow \frac{Z}{\Pi_F} = \frac{E}{\Pi}$ is a convergent of $\frac{X'}{\Pi}$

$\Rightarrow$ rational reconstruction of $X' \mod \Pi$

Correction capacity

Mandelbaum 78:

- 1 symbol = 1 residue
- Polynomial algorithm if $e \leq (n-k)\frac{\log p_{\min} - \log 2}{\log p_{\max} + \log p_{\min}}$
- worst case: exponential (random perturbation)

Goldreich Ron Sudan 99 weighted residues $\Rightarrow$equivalent

Guruswami Sahai Sudan 00 invariably polynomial time

Correction capacity

Mandelbaum 78:

- 1 symbol = 1 residue
- Polynomial algorithm if $e \leq (n-k)\frac{\log p_{\min} - \log 2}{\log p_{\max} + \log p_{\min}}$
- worst case: exponential (random perturbation)

Goldreich Ron Sudan 99 weighted residues $\Rightarrow$equivalent

Guruswami Sahai Sudan 00 invariably polynomial time

Interpretation:

- Errors have variable weights depending on their **impact** $\Pi_F = \prod_{i \in I} p_i$
- Example: $X = 20, p_1 = 2, p_2 = 3, p_3 = 101$
  - 1 error on $X \mod 2$, or $X \mod 3$, can be corrected
  - but not on $X \mod 101$

Plan

Analogy with Reed Solomon

Gao02 Reed-Solomon decoding by extended Euclidean Alg:

- Chinese Remaindering over $K[X]$
- $p_i = X - a_i$
- Encoding = evaluation in a_i
- Decoding = interpolation
- Correction = Extended Euclidean algorithm étendu

Analogy with Reed Solomon

Gao02 Reed-Solomon decoding by extended Euclidean Alg:

- Chinese Remaindering over $K[X]$
- $p_i = X - a_i$
- Encoding = evaluation in a_i
- Decoding = interpolation
- Correction = Extended Euclidean algorithm étendu

$\Rightarrow$Generalization for p_i of degrees > 1
$\Rightarrow$Variable impact, depending on the degree of p_i
$\Rightarrow$Necessary unification [Sudan 01,...]

Plan

Generalized point of view: amplitude code

- Over a Euclidean ring $\mathcal{A}$ with a Euclidean function ν
- Distance

$$\begin{array}{rccc} \Delta: & \mathcal{A} \times \mathcal{A} & \rightarrow & \mathbb{R}_+ \\ & (x, y) & \mapsto & \displaystyle\sum_{i \mid x \neq y [P_i]} \log_2 \nu(P_i) \end{array}$$

Definition

(n,k) amplitude code $C = \{x \in \mathcal{A} : \nu(x) < \kappa\}$,
$n = \log_2 \Pi, k = \log_2 \kappa$.

Generalized point of view: amplitude code

- Over a Euclidean ring $\mathcal{A}$ with a Euclidean function ν
- Distance

$$\begin{array}{rccl} \Delta : & \mathcal{A} \times \mathcal{A} & \rightarrow & \mathbb{R}_+ \\ & (x, y) & \mapsto & \sum_{i \mid x \neq y[P_i]} \log_2 \nu(P_i) \end{array}$$

Definition

(n,k) amplitude code $C = \{x \in \mathcal{A} : \nu(x) < \kappa\}$,
$n = \log_2 \Pi$, $k = \log_2 \kappa$.

Property (Quasi MDS)

$d > n - k$ *in general, and* $d \geq n - k + 1$ *over* $K[X]$.

$\Rightarrow$correction rate = maximal amplitude of an error that can be corrected

Advantages

- Generalization over any Euclidean ring
- Natural representation of the amount of information
- No need to sort moduli
- Finer correction capacities

Advantages

- Generalization over any Euclidean ring
- Natural representation of the amount of information
- No need to sort moduli
- Finer correction capacities
- **Adaptive decoding:** taking advantage of all the available redundancy
- **Early termination:** with no a priori knowledge of a bound on the result

Interpretation of Mandelbaum's algorithm

Remark

Rational reconstruction $\Rightarrow$ *Partial Extended Euclidean Algorithm*

Property

The Extended Euclidean Algorithm, applied to (E, Π) *and to* $(X' = X + E, \Pi)$*, performs the same first iterations until* $r_i < \Pi_V$.

$$\begin{array}{c|c} u_{i-1}E + v_{i-1}\Pi = \Pi_v & u_{i-1}X' + v_{i-1}\Pi = r_{i-1} \\ u_iE + v_i\Pi = 0 & u_iX' + v_i\Pi = r_i \end{array}$$

$$\Rightarrow u_iX = r_i$$

Amplitude decoding, with static correction capacity

Amplitude based decoder over R

Data: Π, X'
Data: $\tau \in \mathbb{R}_+ \mid \tau < \frac{\nu(\Pi)}{2}$: bound on the maximal error amplitude
Result: $X \in R$: corrected message s.t. $\nu(X)4\tau^2 \leq \nu(\Pi)$
begin
  $\alpha_0 = 1, \beta_0 = 0, r_0 = \Pi$;
  $\alpha_1 = 0, \beta_1 = 1, r_1 = X'$;
  $i = 1$;
  while $(\nu(r_i) > \nu(\Pi)/2\tau)$ **do**
    Let $r_{i-1} = q_i r_i + r_{i+1}$ be the Euclidean division of r_{i-1} by r_i;
    $\alpha_{i+1} = \alpha_{i-1} - q_i \alpha_i$;
    $\beta_{i+1} = \beta_{i-1} - q_i \beta_i$;
    $i = i + 1$;
  return $X = -\frac{r_i}{\beta_i}$
end

- reaches the quasi-maximal correction capacity

Amplitude decoding, with static correction capacity

Amplitude based decoder over R

Data: Π, X'

Data: $\tau \in \mathbb{R}_+ \mid \tau < \frac{\nu(\Pi)}{2}$: bound on the maximal error amplitude

Result: $X \in R$: corrected message s.t. $\nu(X)4\tau^2 \leq \nu(\Pi)$

begin

$\alpha_0 = 1, \beta_0 = 0, r_0 = \Pi$;

$\alpha_1 = 0, \beta_1 = 1, r_1 = X'$;

$i = 1$;

while $(\nu(r_i) > \nu(\Pi)/2\tau)$ **do**

Let $r_{i-1} = q_i r_i + r_{i+1}$ be the Euclidean division of r_{i-1} by r_i;

$\alpha_{i+1} = \alpha_{i-1} - q_i \alpha_i$;

$\beta_{i+1} = \beta_{i-1} - q_i \beta_i$;

$i = i + 1$;

return $X = -\frac{r_i}{\beta_i}$

end

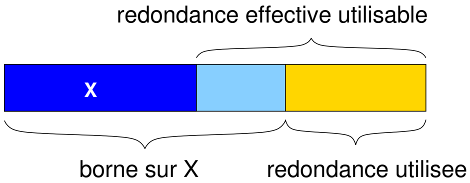
- reaches the quasi-maximal correction capacity
- requires a *a priori* knowledge of τ

$\Rightarrow$ How to make the correction capacity adaptive?

Adaptive approach

Multiple goals:

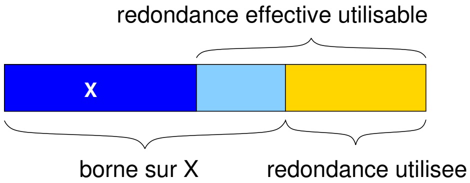
- With a fixed n, the correction capacity depends on a bound on X
  - $\Rightarrow$pessimistic estimate
  - $\Rightarrow$how to take advantage of all the available redundancy?



Adaptive approach

Multiple goals:

- With a fixed n, the correction capacity depends on a bound on X
  $\Rightarrow$pessimistic estimate
  $\Rightarrow$how to take advantage of all the available redundancy?



- Allow early termination: variable n and unknown bound

Plan

A first adaptive approach

Termination criterion in the Extended Euclidean alg.:

- $\alpha_{i+1}\Pi - \beta_{i+1}E = 0$
  $\Rightarrow E = \alpha_{i+1}\Pi/\beta_{i+1}$
  $\Rightarrow$test if β_j divides Π
- check if X satisfies:$\nu(X) \leq \frac{\nu(\Pi)}{4\nu(\beta_j)^2}$
- But several candidates are possible
  $\Rightarrow$discrimination by a post-condition on the result

A first adaptive approach

Termination criterion in the Extended Euclidean alg.:

- $\alpha_{i+1}\Pi - \beta_{i+1}E = 0$
  $\Rightarrow E = \alpha_{i+1}\Pi/\beta_{i+1}$
  $\Rightarrow$test if β_j divides Π
- check if X satisfies:$\nu(X) \leq \frac{\nu(\Pi)}{4\nu(\beta_j)^2}$
- But several candidates are possible
  $\Rightarrow$discrimination by a post-condition on the result

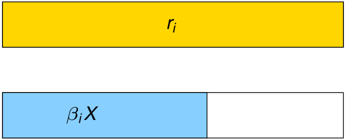
Example

p_i	3	5	7
x_i	2	3	2

- $x = 23$ *with 0 error*
- $x = 2$ *with 1 error*

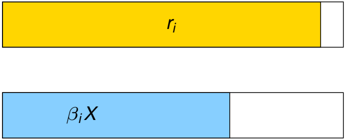
Detecting a gap

$$\alpha_i \Pi - \beta_i (X + E) = r_i \qquad \Rightarrow \qquad \alpha_i \Pi - \beta_i E = r_i + \beta_i X$$



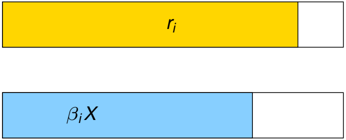
Detecting a gap

$$\alpha_i \Pi - \beta_i (X + E) = r_i \qquad \Rightarrow \qquad \alpha_i \Pi - \beta_i E = r_i + \beta_i X$$



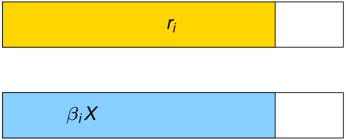
Detecting a gap

$$\alpha_i \Pi - \beta_i (X + E) = r_i \quad \Rightarrow \quad \alpha_i \Pi - \beta_i E = r_i + \beta_i X$$



Detecting a gap

$$\alpha_i \Pi - \beta_i(X + E) = r_i \qquad \Rightarrow \qquad \alpha_i \Pi - \beta_i E = r_i + \beta_i X$$

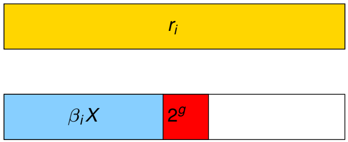


$X = -r_i/\beta_i$

- At the final iteration: $\nu(r_i) \approx \nu(\beta_i X)$
- If necessary, a gap appears between r_{i-1} and r_i.

Detecting a gap

$$\alpha_i \Pi - \beta_i(X + E) = r_i \qquad \Rightarrow \qquad \alpha_i \Pi - \beta_i E = r_i + \beta_i X$$

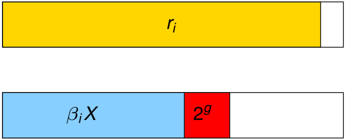


$X = -r_i/\beta_i$

- At the final iteration: $\nu(r_i) \approx \nu(\beta_i X)$
- If necessary, a gap appears between r_{i-1} and r_i.
- $\Rightarrow$Introduce a *blank* 2^g in order to detect a gap $> 2^g$

Detecting a gap

$$\alpha_i \Pi - \beta_i (X + E) = r_i \quad \Rightarrow \quad \alpha_i \Pi - \beta_i E = r_i + \beta_i X$$

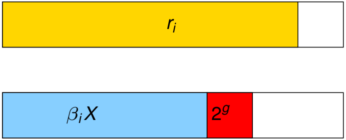


$X = -r_i / \beta_i$

- At the final iteration: $\nu(r_i) \approx \nu(\beta_i X)$
- If necessary, a gap appears between r_{i-1} and r_i.
- $\Rightarrow$Introduce a *blank* 2^g in order to detect a gap $> 2^g$

Detecting a gap

$$\alpha_i \Pi - \beta_i (X + E) = r_i \qquad \Rightarrow \qquad \alpha_i \Pi - \beta_i E = r_i + \beta_i X$$

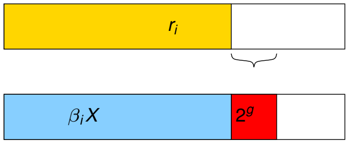


$$X = -r_i / \beta_i$$

- At the final iteration: $\nu(r_i) \approx \nu(\beta_i X)$
- If necessary, a gap appears between r_{i-1} and r_i.
- $\Rightarrow$ Introduce a *blank* 2^g in order to detect a gap $> 2^g$

Detecting a gap

$$\alpha_i \Pi - \beta_i (X + E) = r_i \qquad \Rightarrow \qquad \alpha_i \Pi - \beta_i E = r_i + \beta_i X$$

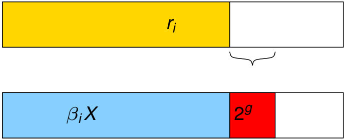


$$X = -r_i / \beta_i$$

- At the final iteration: $\nu(r_i) \approx \nu(\beta_i X)$
- If necessary, a gap appears between r_{i-1} and r_i.
- $\Rightarrow$ Introduce a *blank* 2^g in order to detect a gap $> 2^g$

Detecting a gap

$$\alpha_i \Pi - \beta_i(X + E) = r_i \qquad \Rightarrow \qquad \alpha_i \Pi - \beta_i E = r_i + \beta_i X$$



$X = -r_i/\beta_i$

- At the final iteration: $\nu(r_i) \approx \nu(\beta_i X)$
- If necessary, a gap appears between r_{i-1} and r_i.
- $\Rightarrow$ Introduce a *blank* 2^g in order to detect a gap $> 2^g$

Property

- *Loss of correction capacity: very small in practice*
- *Test of the divisibility for the remaining candidates*
- *Strongly reduces the number of divisibility tests*

Plan

Experiments

Size of the error	10	50	100	200	500	1000
$g = 2$	$1/446$	$1/765$	$1/1118$	$2/1183$	$2/4165$	$1/7907$
$g = 3$	$1/244$	$1/414$	$1/576$	$2/1002$	$2/2164$	$1/4117$
$g = 5$	$1/53$	$1/97$	$1/153$	$2/262$	$1/575$	$1/1106$
$g = 10$	$1/1$	$1/3$	$1/9$	$1/14$	$1/26$	$1/35$
$g = 20$	$1/1$	$1/1$	$1/1$	$1/1$	$1/1$	$1/1$

Table: Number of remaining candidates after the gap detection: c/d means d candidates with a gap $> 2^g$, and c of them passed the divisibility test. $n \approx 6001$ (3000 moduli), $\kappa \approx 201$ (100 moduli).

Experiments

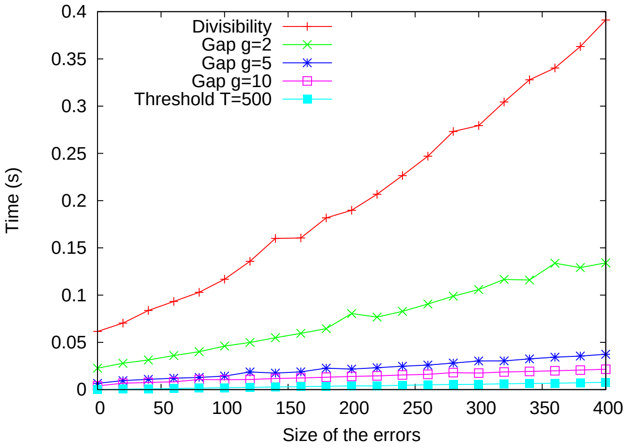


Figure: Comparison for $n \approx 26\,016$ ($m = 1300$ moduli of 20 bits), $\kappa \approx 6001$ (300 moduli) and $\tau \approx 10007$ (about 500 moduli).

Experiments

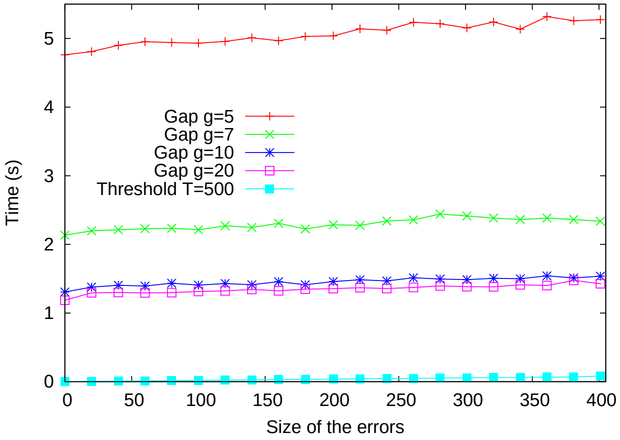


Figure: Comparison for $n \approx 200\,917$ ($m = 10000$ moduli of 20 bits), $\kappa \approx 17\,0667$ (8500 moduli) and $\tau \approx 10498$ (500 moduli).

Gap: Euclidean Algorithm down to the end $\Rightarrow$overhead

Plan

Early termination

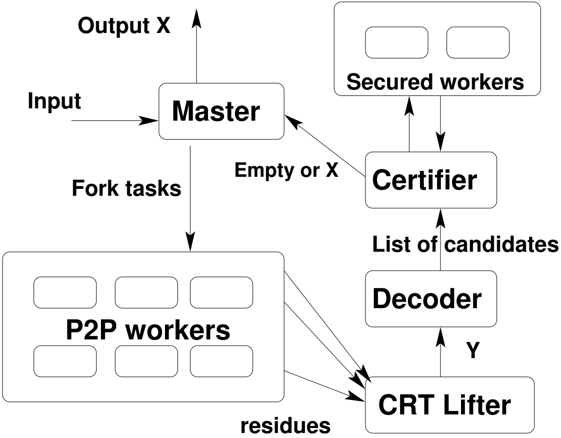


Figure: Fault tolerant distributed computation with early termination

Conclusion

Residue systems : “technology” suited to cloud computing: parallelism, fault-tolerant, verification

New metric for redundant residue codes:

- Unification with finer bounds on the correction capacities
- Enables adaptive decoding

Adaptative decoding and early termination

- Gap method: limited overhead, better performances
- Framework for interactive computing with a cloud

Perspective

- ABFT in exact linear algebra, determinant [LinBox]
- ABFT with (exact) floating point arithmetic
- Theoretical efficiency of the *gap* method,
- Generalization to adaptive list decoding [Sudan, Guruswami]

Jean-Louis Roch's References for this talk

Majid Khonji, Clément Pernet, Jean-Louis Roch, Thomas Roche, and Thomas Stalinski.
Output-sensitive decoding for redundant residue systems.
In *ISSAC '10: Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*, pages 265–272, New York, NY, USA, Jul 2010. ACM.

Axel W. Krings, Jean-Louis Roch, Samir Jafar, and Séebastien Varrette.
A Probabilistic Approach for Task and Result Certification of Large-scale Distributed Applications in Hostile Environments.
In *European Grid Conference EGC'2005 (http://genias.biz/egc2005/)*, Amsterdam, The Netherlands, February 2005. Springer-Verlag, LNCS.

Jean-Louis Roch and Sébastien Varrette.
Probabilistic Certification of Divide & Conquer Algorithms on Global Computing Platforms. Application to Fault-Tolerant Exact Matrix-Vector Product.
In *Parallel Symbolic Computation'07 (PASCO'07)*, London, Ontario, Canada, July 2007. ACM.

Thomas Roche, Jean-Louis Roch, and Matthieu Cunche.
Algorithm-based fault tolerance applied to P2P computing networks.
In *The First International Conference on Advances in P2P Systems*, pages 144–149, Sliema, Malta, Oct 2009. IEEE.