Local dependency dynamic programming in the presence of memory faults

Saverio Caminiti,
Irene Finocchi, and Emanuele G. Fusco

Department of Computer Science, Sapienza University of Rome
Memory fault

• One or more bits is read differently from how were last written
  Hardware problems

• Due to
  Transient electronic noises

• Impact
  Machine crash
  Unpredictable output
  Security vulnerability
How common are memory errors?

- Cluster of 1000 computers
- 4 GB memory each
- One bit error every 3 seconds!
- Each computer: 1 error every 50 minutes

[Schroeder, Pinheiro, and Weber. SIGMETRICS 2009]
Possible Solutions

• Hardware: ECC (not always available)

• Software: robustification
  – Redesign algorithms
  – Rewrite software
  – Faults $\Rightarrow$ longer execution
Faulty RAM model

• Based on the unit cost RAM model
• Adversary
  – Unbounded computational power
  – Can corrupt up to $\delta$ words (at any time)
• $O(1)$ safe memory words
• $O(1)$ private memory words (random bits)

Known results: searching, sorting, dictionaries, priority queues, …

[Finocchi, Italiano, STOC’04]
Local dependency dynamic programming

- Strings $X = x_1 \cdots x_n$ and $Y = y_1 \cdots y_m$ ($n \geq m$)
- $ED(X, Y)$ = the number of edit op {ins, del, sub} required to transform $X$ into $Y$

$$e_{i,j} = \begin{cases} e_{i-1,j-1} & \text{if } x_i = y_j \\ 1 + \min \{e_{i-1,j}, e_{i,j-1}, e_{i-1,j-1}\} & \text{otherwise} \end{cases}$$

- $e_{n,m} = ED(X, Y)$
- $O(nm)$ running time

DP table
A naïf approach

• Resilient variables
  – Write $2\delta+1$ copies
  – Read by majority (in $O(1)$ safe memory)

• Naïf algorithm $O(nm\delta)$ running time

• Match $O(nm)$ running time of the standard non-resilient implementation  \( \Rightarrow \quad \delta = O(1) \)
Algorithm RED (Resilient Edit Distance)

- Assume $X$ and $Y$ are stored resiliently
- $ED(X, Y)$ can be computed:
  - in $O(nm + \alpha \delta^2)$ time
  - $\alpha \leq \delta$ is the actual number of faults
  - correctly $w.h.p.$
- Assume $m = \Theta(n)$:
  - match $O(n^2)$ \implies $\delta = O(n^{2/3})$
Techniques

• Resilient variables

• Table decomposition (one-level/hierarchical)

• Karp-Rabin fingerprints
  – Can be computed incrementally in $O(1)$ private memory

• Partial recomputation upon fault detection
Table decomposition

• DP table is split into blocks of $\delta \times \delta$ cells

• Last row and column are written reliably in the unreliable memory
Block computation

• Column-major order
• While writing column $h$
  compute write fingerprint $\varphi_h$
on written data
• While reading column $h$
  compute read fingerprint $\gamma_h$
on read data
• Fingerprint test:
  if $\varphi_h \neq \gamma_h$ recompute block
• Similar fingerprints for $X$ and $Y$
Running time analysis

• **Successful** block computations:
  – No fingerprint mismatch
  – $O(1)$ amortized cost per operation $\Rightarrow O(nm)$

• **Unsuccessful** block computations:
  – Each block recomputation can be attributed to (at least) a distinct fault
  – $\alpha$ faults $\Rightarrow O(\alpha \delta^2)$

• Overall running time: $O(nm + \alpha \delta^2)$

• Correct w.h.p. (game based proof)
Tracing back

• Edit sequence is given by $\pi$
• In each block traversed by $\pi$
  – Compute a segment of $\pi$ unreliable
  – Verify the segment reading input and block borders reliably
  – Segment not valid $\implies$ recompute the block forward
Faster error recovery

• Edit distance and sequence can be computed:
  • in $O(nm + \alpha\delta^{1+\varepsilon})$ time
  • correctly w.h.p.

• Assume $m = \Theta(n)$:
  
  match $O(n^2) \implies \delta = O(n^{2/(2+\varepsilon)})$
Semi-resilient data

• An $r$–resilient variable
  – written in $2r+1$ copies and read by majority
  – can be corrupted (as $r < \delta$) but at the cost of $> r$ faults

• $k$ resiliency levels ($k$ constant $= 1/\varepsilon$)
  – level $i \in [1,k]$ uses on $\delta_i$–resilient variables, $\delta_i = \lceil \delta^{i/k} \rceil$

E.g., with $k = 3$

- $\delta^{1/3}$ –resilient
- $\delta^{2/3}$ –resilient
- $\delta$–resilient
Long-distance fingerprints

• Every $\delta_i$ columns we store a $\delta_i$–resilient copy

• One fingerprint for resiliency level ($k$ fingerprints)

• Level $i$ fingerprint associated with the last column written $\delta_i$–resilient
Long-distance fingerprints

• Fingerprint mismatch on non resilient columns:
  – restart computation from the last $\delta_1$–resilient column

• Fingerprint mismatch while reading at level $i$:
  – restart computation from the last $\delta_{i+1}$–resilient column
Trace-back with semi-resilient cols

- Exploit semi-resilient columns but intermediate fingerprints are no longer available

- Compute segments at resiliency level $i$ and glue them together to obtain segments at level $i+1$
Trace-back with semi-resilient cols

- Level $i$ segments are **verified** against $\delta_i$–resilient columns

- Invalid segment $\Rightarrow$ recompute forward only the $\delta_{i/k}$ slice of the DP table

$$O(nm + \alpha \delta^{1+\varepsilon})$$
Conclusions

• All Local Dependency Dynamic Programming problems

• Generalize to higher dimensions

• Well known optimization techniques:
  – Hirschberg: reduce space usage
  – Ukkonen: reduce running time if strings are similar
The End