Regular Expression Search on Compressed Text

without the need for decompression

Javier Esparza, Technical University of Munich
Pedro Valero & Pierre Ganty, IMDEA Software Institute
https://pevalme.github.io

Problem

Big Data requires handling gazillions of data

To save money. . .

Tech giants keep information compressed in their data centers

Develop compression engines. . .

`brotli` (*Google*), `zstd` (*Facebook*) . . .

But they need to process the data. . .

Regular expression search is the most basic operation

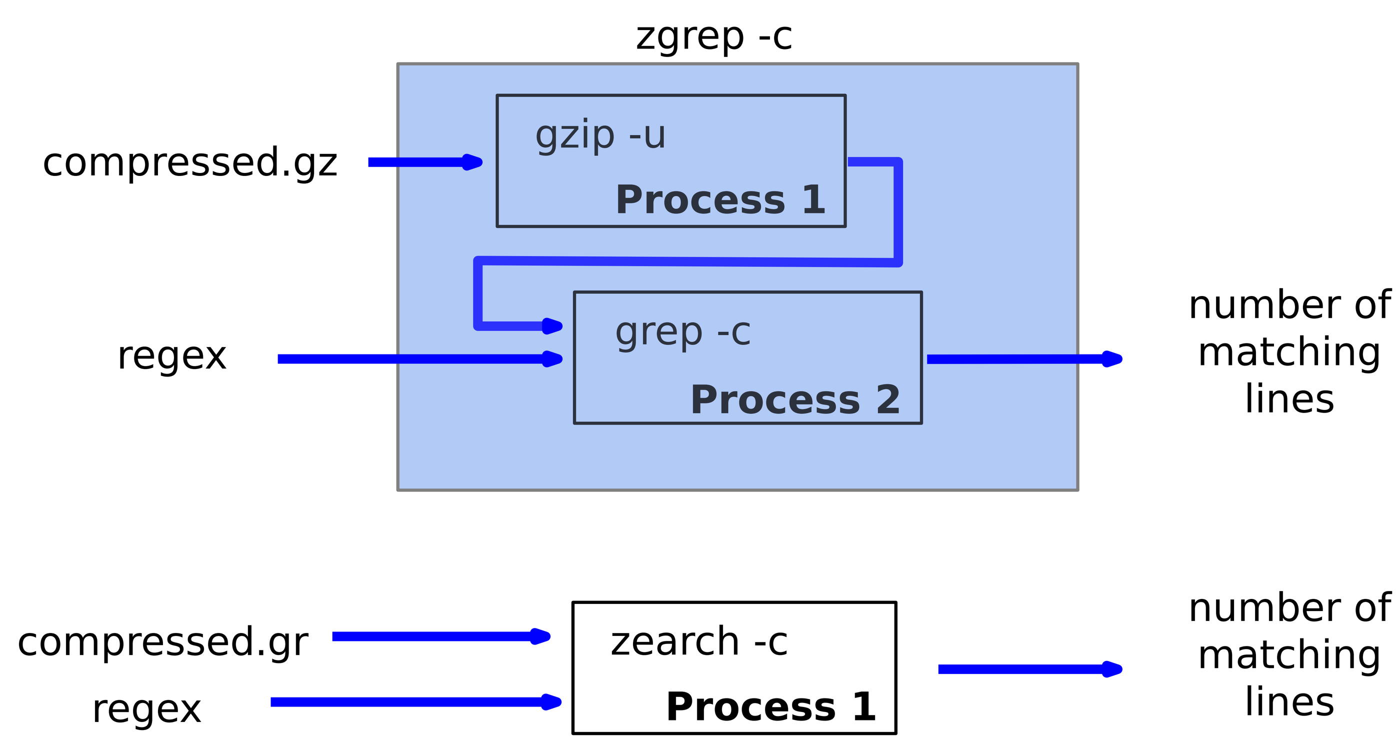
Develop regex engines. . .

RE2 (*Google*), `Hyperscan` (*Intel*) . . .

But when searching in compress data. . .

`zgrep = gzip -d | grep`

State of the art vs *zearch*



Overview of *zearch*

Setup:

Data is compressed with a **grammar-based** algorithm (LZ78, LZW, Sequitur, Repair. . .)
Regular Expression is used tu build a **non-deterministic** automata without ϵ-transitions

Idea:

- **Use redundancies to speed up the search.**

  Each rule is processed *only* once

  Processing n rules may cover 2^n characters: $\{X_0 \to X_1X_1\} \cup \{X_i \to X_{i+1}X_{i+1} \mid i = 1 \dots n\}$

- **Saturation construction**

  $p \xrightarrow{a} q$ NFA moves from p to q when reading a

  $p \xrightarrow{X} q$ NFA moves from p to q when reading $\mathcal{L}(X)$

Algorithm:

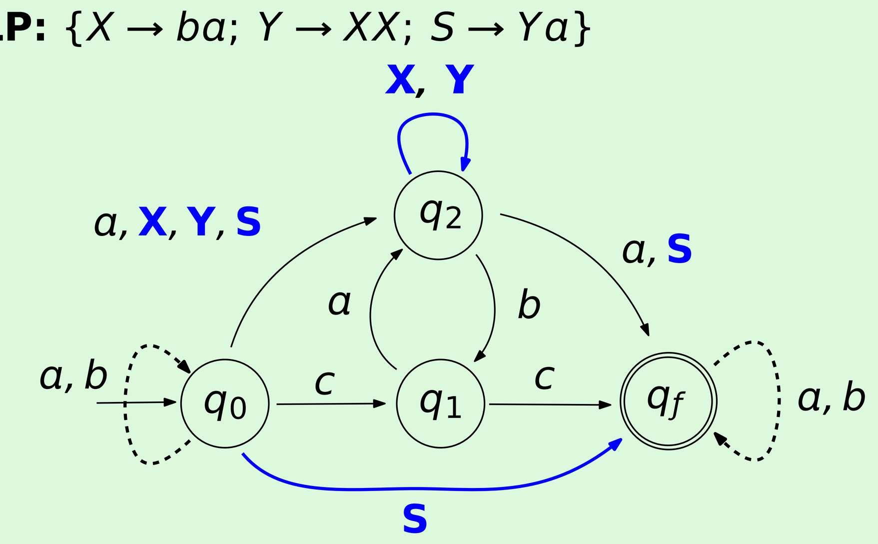
Input: SLP $G = (\Sigma, V, \{X_1 \to \alpha_1\beta_1, \dots X_n \to \alpha_n\beta_n\})$
Automata $A = (\Sigma, Q, q_0, F, \delta)$

Output: Saturated automata $A^s = (\Sigma, Q, q_0, F, \delta^s)$

For each $(X_i \to \alpha_i\beta_i) \in \{X_1 \to \alpha_1\beta_1, \dots X_n \to \alpha_n\beta_n\}$
  For each $(q_1, \alpha, q') \in \delta$
    For each $(q', \beta, q_2) \in \delta$
      $\delta := \delta \cup \{(q_1, X, q_2)\}$
      INCR_COUNT(X, q')

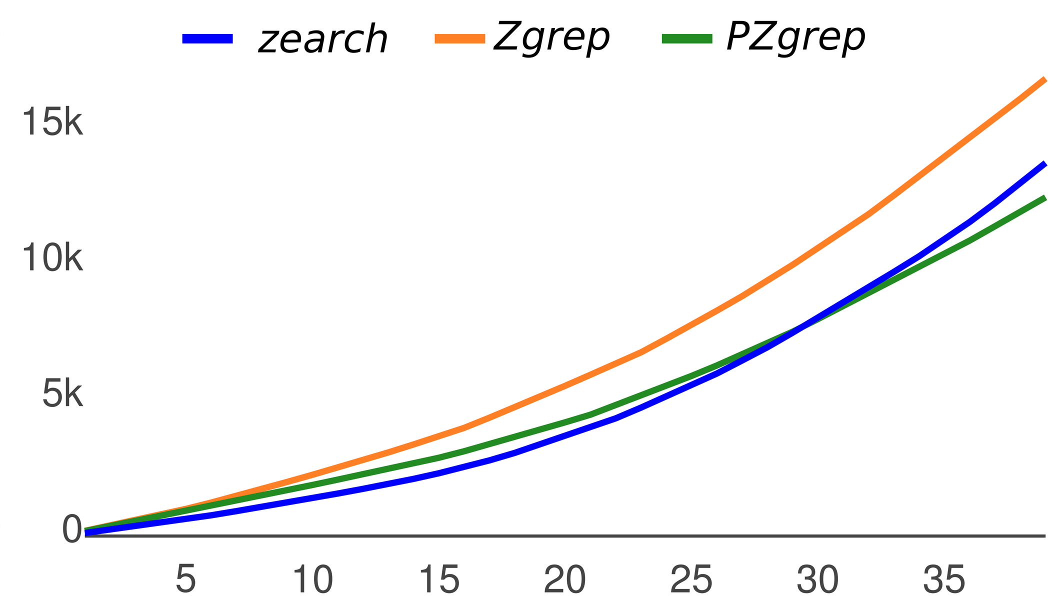
Example:

SLP: $\{X \to ba;\ Y \to XX;\ S \to Ya\}$



Experimental Results

	Log (100MB → 7 MB) zearch	Zgrep	PZgrep	JSON (100MB → 8.8 MB) zearch	Zgrep	PZgrep	Subtítulos (100MB → 14 MB) zearch	Zgrep	PZgrep	CSV (100MB → 27 MB) zearch	Zgrep	PZgrep
“pedro”	**110**	261	212	**152**	280	211	242	305	**236**	505	417	**377**
“.”	**130**	227	197	**158**	185	194	283	367	**273**	595	529	**403**
“I .∗ you”	**130**	272	199	**210**	290	213	391	384	**277**	541	373	**351**
“[a-z]{4}”	**161**	270	206	278	188	**186**	407	494	**354**	694	621	**479**
“ [a-z]{4} ”	**136**	409	262	**177**	229	212	**398**	695	519	568	712	**473**
“[a-z]∗[a-z]{4}”	**172**	266	205	324	191	**188**	481	521	**399**	755	644	**462**
“[a-z]∗[a-z]{6}”	**192**	302	222	396	194	**195**	539	596	**436**	819	650	**480**
“ \d5\d0\d4\d5\d ”	**123**	416	269	**153**	272	205	**309**	705	527	608	724	**510**
“([a-z]{5})+”	**134**	410	262	**206**	235	210	614	706	**531**	575	727	**512**
“([a-z]{5} +){5}”	**174**	528	385	**323**	545	404	652	751	**583**	744	748	**468**
Average	**146**	336	241	237	260	**221**	431	552	**413**	640	614	**451**



Better compression ⟹ More repetitions in the data ⟹ Less grammar rules to be processed ⟹ *zearch* is faster

Data Structure

$X_1 \to a_0a_1$ $X_2 \to a_3a_4$ $X_3 \to X_2X_1$

$X \to \sigma$, $Y \to \rho$ are independent iff $X \notin \rho \wedge Y \notin \sigma$
Independent rules can be processed simultaneously
Conceptually easy parallelization