Compressing Wikipedia

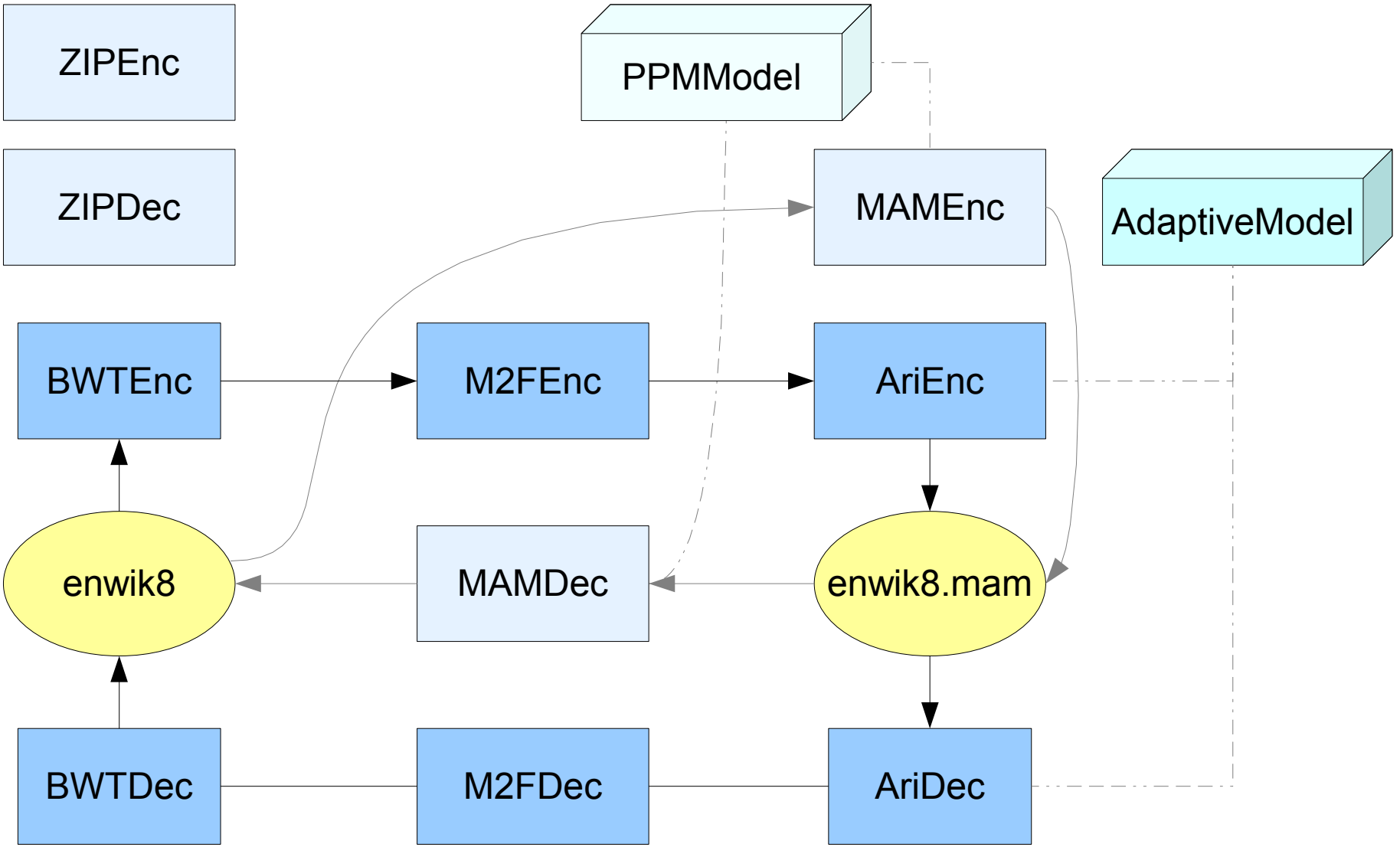
Group #1

Aija Niissalo
Marko Laakso
Markus Heinonen

Stream Transformation Framework

- Generic framework that converts standard input to standard output
- Pipeline architecture with accessible intermediate values
- Common interface for all transformations
- The basic functionality that is needed for a stand-alone application
- Set of useful stream handling utilities
- Implemented in Java and tested with JUnit

Encoding Pipeline



Burrows-Wheeler Transformation

- Reversible permutation of the characters
- No compression (extra bytes for the starting position)
- Long sequences of repetitive strings
  - Local entropy becomes lower
- Implementation was based on Wikipedia
  - C-program was converted into Java
  - Quick sort replaced with merge sort
  - Some optimizations
  - Block size of 500KB

Burrows-Wheeler Example

Input	Rotations	Sorted rotations	Output
	^BANANA@	ANANA@^B	
	@^BANANA	ANA@^BAN	
	A@^BANAN	A@^BANAN	
^BANANA@	NA@^BANA	BANANA@^	BNN^AA@A
	ANA@^BAN	NANA@^BA	
	NANA@^BA	NA@^BANA	
	ANANA@^B	^BANANA@	
	BANANA@^	@^BANANA	

This example has been taken from Wikipedia

Move-to-front Transformation

- Reversible encoding of the character indices
- No compression, block based transformation
  - May compress if symbols are longer than indices
- Each input symbol (we use bytes) has an index
- Each input symbol is replaced with its index and the indices are shifted so that the index of the symbol becomes 0
- Repetitive sequences are encoded with small indices thus the distribution of output symbols is highly skewed

Arithmetic Coding

- A message is represented by a real interval [a,b) from the base interval [0,1)
- The more likely symbols reduce the range by less than the unlikely symbols → fewer bits to message
- Each symbol s_i has its own interval $[a_i,b_i)$ such that p_i is/(is proportional to) $b_i - a_i$
- Problem 1 = need of infinite precision
- Solution 1 = use integer arithmetics [0,1) → e.g. $[0,2^{32})$
- → Problem 2 = consecutive symbol readings → interval shrinks into a single integer
- Solution 2 = use rescaling

Arithmetic Coding – steps

Encoding

1. set [a,b) := [0,1)
2. step 3 to all s_i
3. [a,b) := [a+(b-a)*a_i, a+(b-a)*b_i)

Decoding code = [c,d)

1. set [a,b) := [0,1)
2. - set x := (c-a)/(b-a)

   - find i such that a_i = x < b_i

   - print s_i
3. update interval:
   [a,b) := [a+(b-a)*a_i, a+(b-a)*b_i)
4. repeat steps 2 and 3 until 'end'-symbol is reached

Arithmetic Coding - example

si	pi	range [0, 1)
a	0.2	[0, 0.2)
e	0.3	[0.2, 0.5)
w	0.1	[0.5, 0.6)
o	0.2	[0.6, 0.8)
u	0.1	[0.8, 0.9)
'end'	0.1	[0.9, 1)

Arithmetic Coding - example

initial	range [0, 1)
read e	[0.2, 0.5)
read o	[0.38, 0.44)
read e	[0.392, 0.41)
read 'end'	[0.4082, 0.41)

While decoding variable x has values 0.4082, 0.694, 0.47 and 0.9, corresponding symbols: e, o ,e and 'end'

In practice the code is some number from the code interval e.g. 0.409

Character Frequency Model

Adaptive model

- Start with uniform distribution
- Update model for each symbol
- Re-scaling before overflow

Possible improvements

- Start with a skewed distribution
- Sliding window
- Prediction with partial matching

Prediction by partial matching PPM

- Adaptive statistical modeling technique based on blending together different length context models to predict the next character in the input sequence
- If no prediction can be made based on all n context symbols a prediction is attempted with just n-1 symbols

PPM implementation

In the implementation:

- set the initial probabilities of all of the symbols to 0 for a given context, and have a method to fall-back to a different context when a previously unseen symbol occurs. This is done by emitting a special code, referred to as an Escape code.
- e.g. the depth of the tree = n

- The -1 context is set up at initialization to have a count of 1 for every possible symbol, and doesn't ever get updated. So it is guaranteed to be able to encode every symbol.

Results

- Generic compression platform with several transformations implemented
  - Tested and documented API
- Compression of enwik8
  - Native executable with multiprocessor support
  - Size: 45779 (program) + 34066116 (enwik8.mam) = 34111895
  - Requirements: RAM < 60MB, time < 7min