January 2016

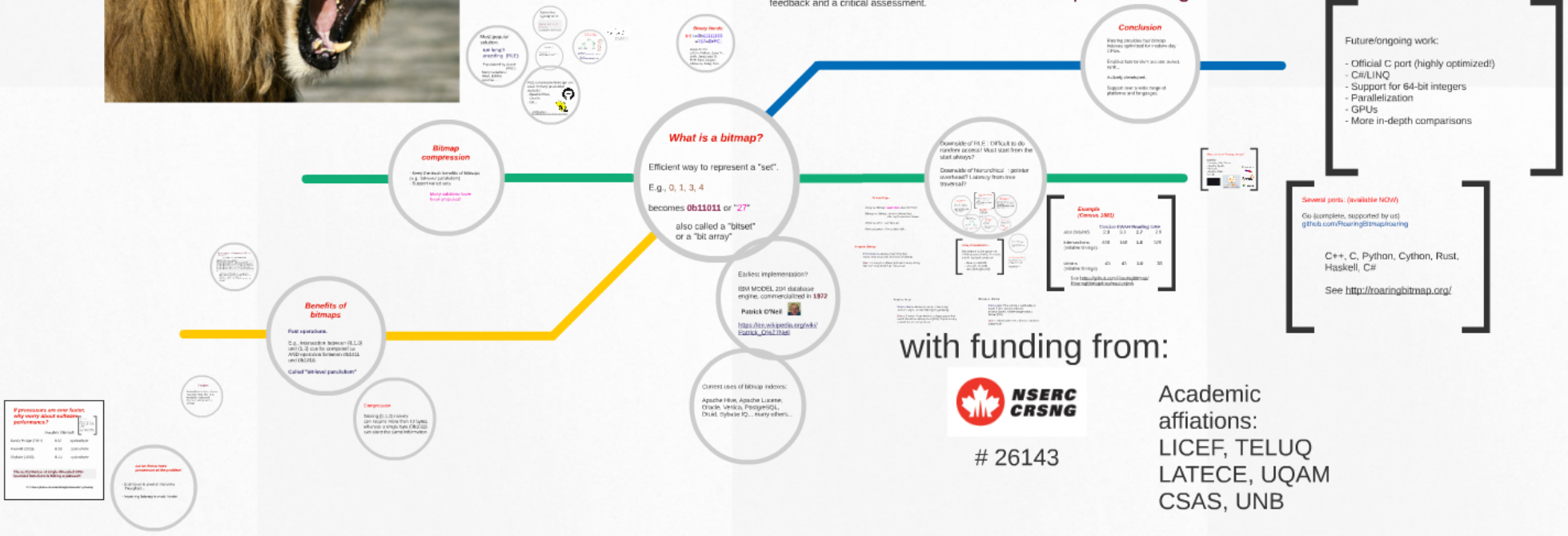
http://roaringbitmap.org/
https://github.com/RoaringBitmap

Roaring Bitmaps

Consistently faster and smaller compressed bitmaps with Roaring (in preparation)

Better bitmap performance with Roaring bitmaps, Software: Practice and Experience (to appear). http://onlinelibrary.wiley.com/doi/10.1002/spe.2325/abstract
http://arxiv.org/abs/1402.6407

The Roaring project benefited from so many contributions that it is not possible to provide an exhaustive list. Among others, we would like to acknowledge software contributions from J. Alvarado, S. Chambi, W. Glynn, R. Graves, T. Gruben, B. Ivanov, T. Maly, E. Murphy, F. Saint-Jacques, S. Pellegrino, B. Potter, G. Punti, B. Sperber. We are also grateful to the **Druid developers** (including C. Allen and F. Yang) for their help and feedback, and would like to thank X. Léauté specifically for providing test cases that motivated our work on run containers. The **Apache Spark developers** (including D. Liu, S. Owen, I. Rashid, R. Xin and K. Yao) and **Apache Kylin developers** (including L. Han, L. Yang) provided feedback and a critical assessment.



with funding from:

NSERC CRSNG

26143

Academic affiations:
LICEF, TELUQ
LATECE, UQAM
CSAS, UNB

January 2016

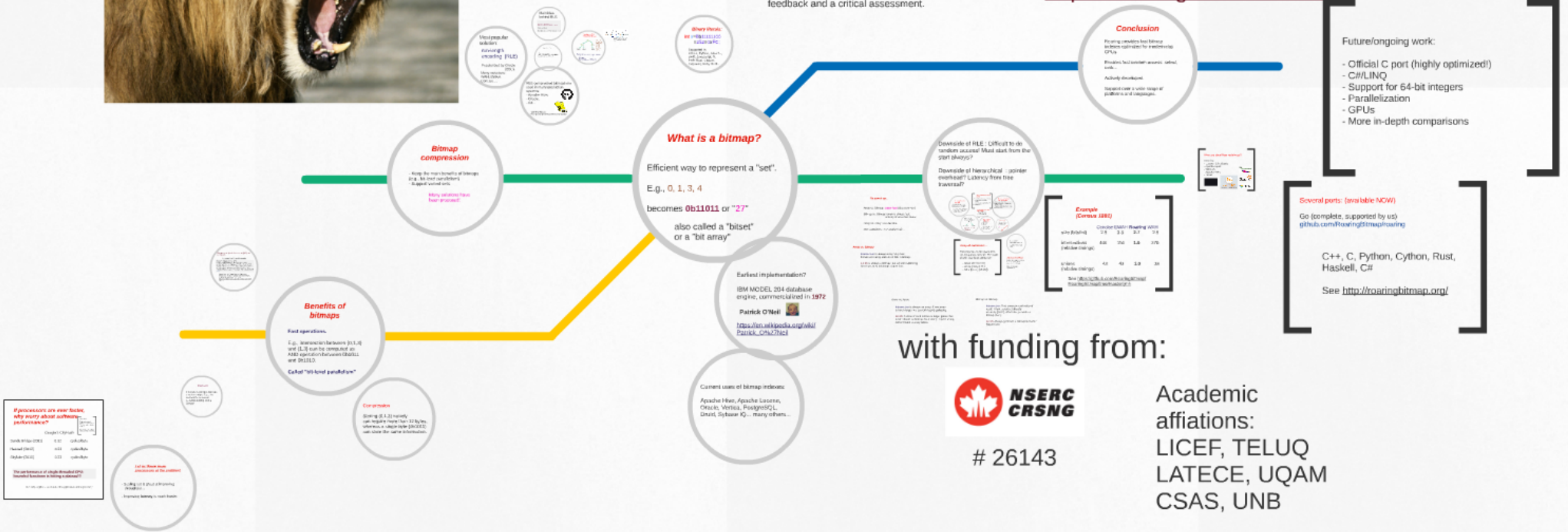
http://roaringbitmap.org/
https://github.com/RoaringBitmap

Roaring Bitmaps

Consistently faster and smaller compressed bitmaps with Roaring (in preparation)

Better bitmap performance with Roaring bitmaps, Software: Practice and Experience (to appear).
http://onlinelibrary.wiley.com/doi/10.1002/spe.2325/abstract
http://arxiv.org/abs/1402.6407

The Roaring project benefited from so many contributions that it is not possible to provide an exhaustive list. Among others, we would like to acknowledge software contributions from J. Alvarado, S. Chambi, W. Glynn, R. Graves, T. Gruben, B. Ivanov, T. Maly, E. Murphy, F. Saint-Jacques, S. Pellegrino, B. Potter, G. Puntí, B. Sperber. We are also grateful to the **Druid developers** (including C. Allen and F. Yang) for their help and feedback, and would like to thank X. Léauté specifically for providing test cases that motivated our work on run containers. The **Apache Spark developers** (including D. Liu, S. Owen, I. Rashid, R. Xin and K. Yao) and **Apache Kylin developers** (including L. Han, L. Yang) provided feedback and a critical assessment.



with funding from:

NSERC CRSNG

26143

Academic affiations:
LICEF, TELUQ
LATECE, UQAM
CSAS, UNB

Why care about Roaring bitmaps?

Used by:

- Lucene, Solr, Elastic
- Apache Spark,
- Whoosh,
- Apache Kylin,
- Druid.

If processors are ever faster, why worry about software performance?

This means:

"High performance" programming will remain very important in coming decade.

We have to tune software for the hardware we have...

Google's CityHash

Sandy Bridge (2011)	0.32	cycles/byte
Haswell (2013)	0.23	cycles/byte
Skylake (2015)	0.23	cycles/byte

The performance of single-threaded CPU-bounded functions is hitting a plateau!!!

Ref: https://github.com/lemire/StronglyUniversalStringHashing

Binary literals:

```
int x=0b11111100
    =252=0xFC;
```

Supported in
C/C++, Python, Java 7+,
Swift, JavaScript, D,
PHP, Rust, Clojure,
C#(soon), Ruby, Perl...

What is a bitmap?

Efficient way to represent a "set".

E.g., 0, 1, 3, 4

becomes **0b11011** or "27"

also called a "bitset"
or a "bit array"

"bit array"

Earliest implementation?

IBM MODEL 204 database engine, commercialized in **1972**

Patrick O'Neil

https://en.wikipedia.org/wiki/Patrick_O%27Neil

ersections,
2/spe.2289/

Benefits of bitmaps

Fast operations.

E.g., intersection between {0,1,3} and {1,3} can be computed as AND operation between 0b1011 and 0b1010.

Called "bit-level parallelism"

Compression

Storing {0,1,3} naively
can require more than 12 bytes,
whereas a single byte (0b1011)
can store the same information.

Problem:

If density is too low, bitmaps lose their edge. E.g., it is wasteful to represent {1,32000,64000} with a bitmap!

Bitmap compression

- Keep the main benefits of bitmaps (e.g., bit-level parallelism)
- Support varied sets

Many solutions have been proposed!

Most popular solution:

run-length encoding (RLE)

Popularized by Oracle (BBC).

Many variations: WAH, EWAH, Concise.....

RLE-con
used in r

acle
BC).

RLE-compressed bitmaps are used in many production systems:
- Apache Hive,
- Oracle,
- Git...

Counting Objects
http://githubengineering.com/counting-objects/

Main idea behind RLE:

0b0000....000000011111.....11111

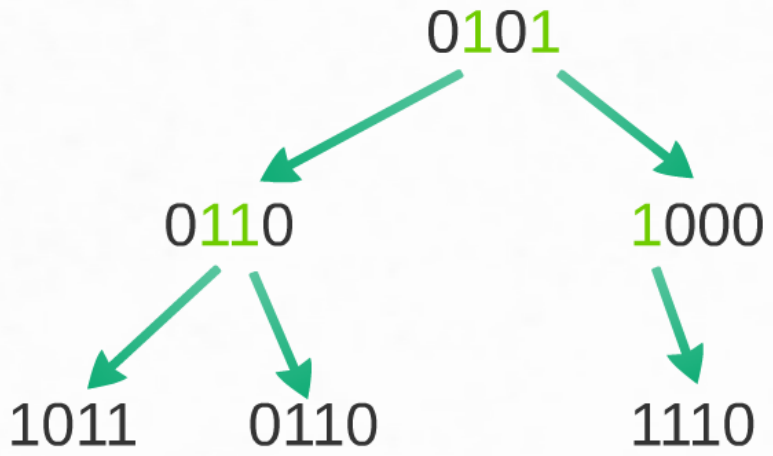
Then code it as

<number of zeros> <number of ones>

Benefit of RLE:

- Good compression
- Pretty fast decoding (though limited by branch mispredictions?)

Variation on RLE: hierarchical bitmaps



This is the compressed representation of a 4 x 4 x 4-bit bitmap containing many zeroes

SparseBitSet:
https://github.com/brettwooldridge/SparseBitSet

Bitmagic:
http://bmagic.sourceforge.net/

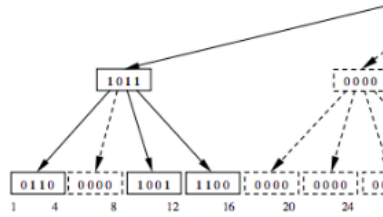
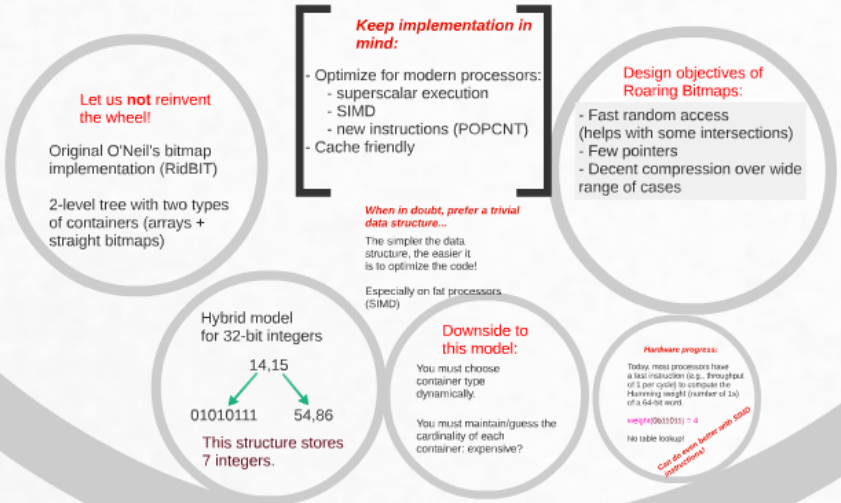


Fig. 1. A sing

Downside of RLE : Difficult to do random access! Must start from the start always?

Downside of hierarchical : pointer overhead? Latency from tree traversal?



also common)

ways fast,
rsection slower

alk...

siz

New hardware +

Prezi

on in

essors:

CNT)

rivial

rs

Design objectives of Roaring Bitmaps:

- Fast random access (helps with some intersections)
- Few pointers
- Decent compression over wide range of cases

Keep implementation in mind:

- Optimize for modern processors:
  - superscalar execution
  - SIMD
  - new instructions (POPCNT)
- Cache friendly

When in doubt, prefer a trivial data structure...

The simpler the data structure, the easier it is to optimize the code!

Especially on fat processors (SIMD)

Array of containers...

Decompose 32-bit space into 16-bit spaces (chunk). For each chunk, use best container:

- bitset (01001011)
- array ({1,20,144})
- runs ([0,10],[15,20])

Especially on fat processors (SIMD)

Downside to this model:

You must choose container type dynamically.

You must maintain/guess the cardinality of each container: expensive?

Hardware progress:

Today, most processors have a fast instruction (e.g., throughput of 1 per cycle) to compute the Hamming weight (number of 1s) of a 64-bit word.

weight(0b11011) = 4

No table lookup!

Can do even better with SIMD instructions!

New hardware +
well chosen algorithms

make the hybrid model
fast.

Implementation challenge

Instead of making everything fit in one model (bitset or array), we mix them.

Many optimization opportunities!

Bitmap vs. Bitmap

Intersection: First compute cardinality of result, if low, construct directly an array (pain!), otherwise generate a bitmap (fast).

Union: Always generate a bitmap container (super fast)

Array vs. Array

Intersection is always an array. If one array is much larger, we use O(n log m) galloping.

Union: if some of cardinalities is large, guess that result should be bitmap. Do it (fast). If got it wrong, convert back to array (slow).

Array vs. Bitmap

Intersection is always array. Very fast:
iterate over array and check bits in bitmap.

Union is always a bitmap: just set corresponding bits from array in bitmap. Super fast.

To sum it up...

Array vs. Bitmap: **super fast** (also common)

Bitmap vs. Bitmap: union is always fast,
unlucky intersection slowe

Array vs. Array : can be slow

Run containers : For another talk...

Example (Census 1881)

	Concise	EWAH	**Roaring**	WAH
size (bits/int)	2.9	3.3	**2.7**	2.9
intersections (relative timings)	460	150	**1.0**	370
unions (relative timings)	43	43	**1.0**	38

See https://github.com/RoaringBitmap/RoaringBitmap/tree/master/jmh

Several ports: (available NOW)

Go (complete, supported by us)
github.com/RoaringBitmap/roaring

C++, C, Python, Cython, Rust, Haskell, C#

See http://roaringbitmap.org/

Future/ongoing work:

- Official C port (highly optimized!)
- C#/LINQ
- Support for 64-bit integers
- Parallelization
- GPUs
- More in-depth comparisons

Conclusion

Roaring provides fast bitmap indexes optimized for modern-day CPUs.

Enables fast random access: select, rank...

Actively developed.

Support over a wide range of platforms and languages.

January 2016

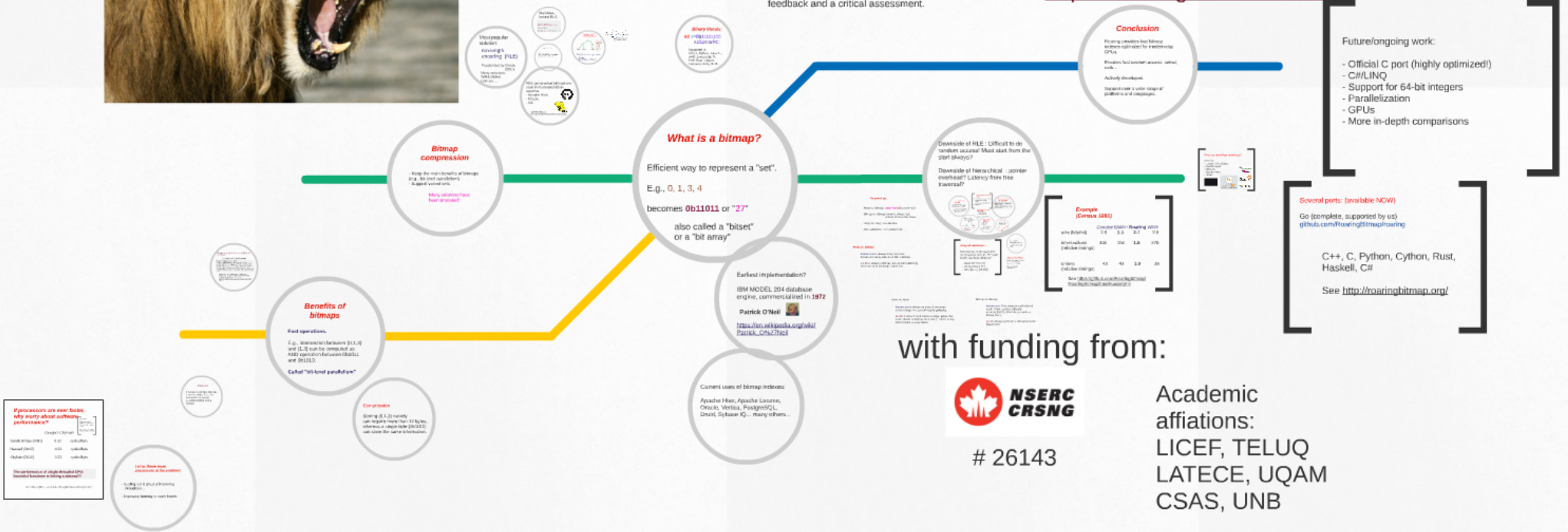
http://roaringbitmap.org/
https://github.com/RoaringBitmap

Roaring Bitmaps

Consistently faster and smaller compressed bitmaps with Roaring (in preparation)

Better bitmap performance with Roaring bitmaps, Software: Practice and Experience (to appear).
http://onlinelibrary.wiley.com/doi/10.1002/spe.2325/abstract
http://arxiv.org/abs/1402.6407

The Roaring project benefited from so many contributions that it is not possible to provide an exhaustive list. Among others, we would like to acknowledge software contributions from J. Alvarado, S. Chambi, W. Glynn, R. Graves, T. Gruben, B. Ivanov, T. Maly, E. Murphy, F. Saint-Jacques, S. Pellegrino, B. Potter, G. Puntí, B. Sperber. We are also grateful to the **Druid developers** (including C. Allen and F. Yang) for their help and feedback, and would like to thank X. Léauté specifically for providing test cases that motivated our work on run containers. The **Apache Spark developers** (including D. Liu, S. Owen, I. Rashid, R. Xin and K. Yao) and **Apache Kylin developers** (including L. Han, L. Yang) provided feedback and a critical assessment.



with funding from:

NSERC CRSNG

26143

Academic affiations:
LICEF, TELUQ
LATECE, UQAM
CSAS, UNB