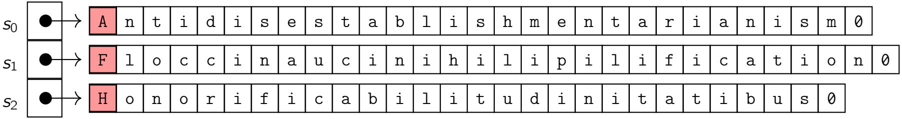
Karlsruhe Institute of Technology

# Communication-Efficient String Sorting

Timo Bingmann, Peter Sanders, Matthias Schimek · 2020-05-18 @ IPDPS’20

INSTITUTE OF THEORETICAL INFORMATICS – ALGORITHMICS



$s_0$ A n t i d i s e s t a b l i s h m e n t a r i a n i s m 0

$s_1$ F l o c c i n a u c i n i h i l i p i l i f i c a t i o n 0

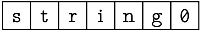
$s_2$ H o n o r i f i c a b i l i t u d i n i t a t i b u s 0

Video and More Information:

https://panthema.net/2020/0518-distributed-string-sorting/

# Why String Sorting?

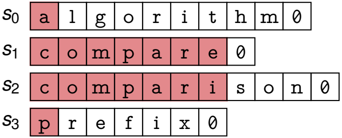
- string: array of characters over alphabet $\Sigma$



- sorted string set: sorted lexicographically
  $\Rightarrow$ like in a dictionary

- characteristics of string sets
  - #strings $n$, #characters $N$
  - sum distinguishing prefix lengths $D$

  $\Rightarrow$ multidimensional data



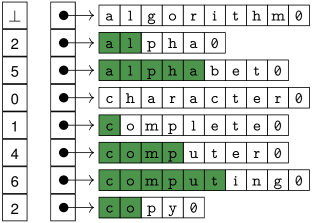
- only published distributed string sorting algorithm: one paragraph in [Fischer and Kurpicz, ALENEX'19]

# String Sorting Toolbox

- **Sequential Sorting:** String Radix Sort, Multikey Quicksort, . . .

  [Kärkkäinen et al., SPIRE'08], [Bentley and Sedgewick, SODA'97]

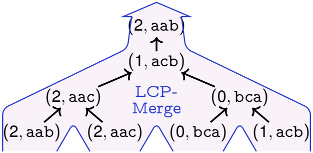
  - evaluation of many sequential algorithms in [Bingmann '18]
  - needed: string sorting + Longest Common Prefix (LCP) array computation



- **Multiway Merging:** **LCP Losertree**

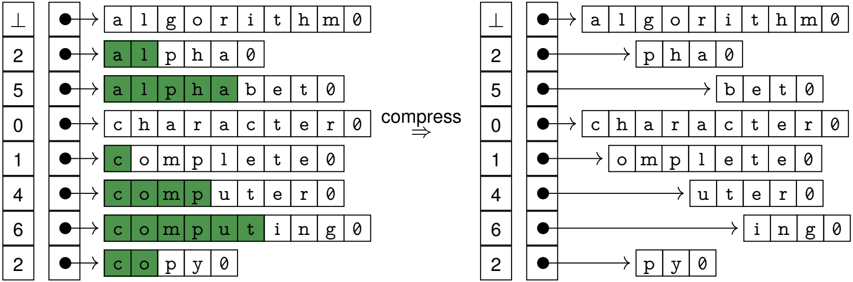
  [Bingmann et. al, Algorithmica'17]

  - exploit LCP values to save character-comparisons



# String Sorting Toolbox

## LCP Compression

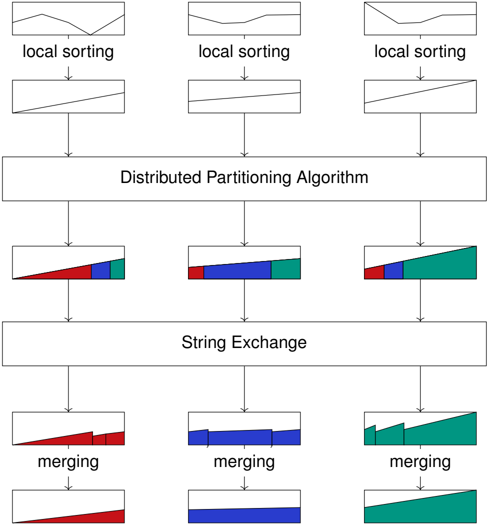


| | |
|---|---|
| ⊥ | a l g o r i t h m 0 |
| 2 | a l p h a 0 |
| 5 | a l p h a b e t 0 |
| 0 | c h a r a c t e r 0 |
| 1 | c o m p l e t e 0 |
| 4 | c o m p u t e r 0 |
| 6 | c o m p u t i n g 0 |
| 2 | c o p y 0 |

| | |
|---|---|
| ⊥ | a l g o r i t h m 0 |
| 2 | p h a 0 |
| 5 | b e t 0 |
| 0 | c h a r a c t e r 0 |
| 1 | o m p l e t e 0 |
| 4 | u t e r 0 |
| 6 | i n g 0 |
| 2 | p y 0 |

- each longest common prefix is sent only once
- compression: iterate over strings + LCP array
- decompression: iterate over compressed strings + LCP array

# Distributed Merge String Sort (MS)



- **Local Sorting**
  - String Radix Sort

  new: String Radix Sort + LCP array

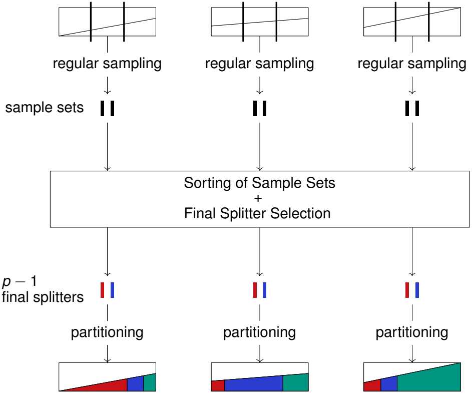
- **String Exchange**
  - no compression

  new: LCP compression

- **Merging**
  - plain losertree

  new: LCP losertree

# Distributed Merge String Sort (MS)



- **Partitioning**
  - equidistant sampling
  - gather + seq. sort

  new: hypercube quicksort [Axtmann and Sanders, ALENEX'17]
  - broadcast final splitters
  - partitioning

# Prefix Doubling String Merge Sort (PDMS)

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PE1: | A | n | t | i | d | i | s | e | s | t | a | b | l | i | s | h | m | e | n | t | a | r | i | a | n | i | s | m | 0 | |
| PE2: | F | l | o | c | c | i | n | a | u | c | i | n | i | h | i | l | i | p | i | l | i | f | i | c | a | t | i | o | n | 0 |
| PE3: | H | o | n | o | r | i | f | i | c | a | b | i | l | i | t | u | d | i | n | i | t | a | t | i | b | u | s | 0 | | |

- same main structure as before

- use distributed Single-Shot Bloom Filter (dSBF) to approximate distinguishing prefixes with distributed duplicate detection [Sanders et al., IEEE BigData'13]

- only operate on those characters

- calculate only the permutation for sorting (exchanging further characters is optional).

# Experimental Evaluation – Setup

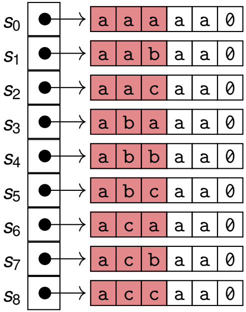
## Input Data

- weak scaling with *D*/*N*-Generator

## Hardware (ForHLR I at KIT)

- 2 Deca-core Intel Xeon E5-2670 v2 (2.5 GHz) and
- 64 GB RAM per compute node
- InfiniBand 4X FDR interconnect

## *D*/*N*-Generator
## (*n*=9, *ℓ*=6, *D*/*N*=0.5)

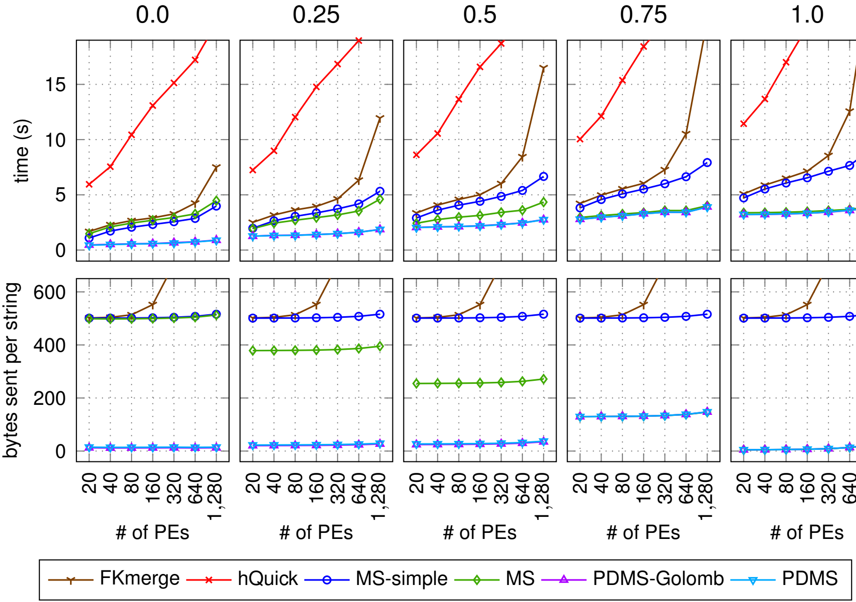


| | | | | | | |
|---|---|---|---|---|---|---|
| $s_0$ | a | a | a | a | a | 0 |
| $s_1$ | a | a | b | a | a | 0 |
| $s_2$ | a | a | c | a | a | 0 |
| $s_3$ | a | b | a | a | a | 0 |
| $s_4$ | a | b | b | a | a | 0 |
| $s_5$ | a | b | c | a | a | 0 |
| $s_6$ | a | c | a | a | a | 0 |
| $s_7$ | a | c | b | a | a | 0 |
| $s_8$ | a | c | c | a | a | 0 |

## Algorithms

- FKmerge: from Fischer and Kurpicz [ALENEX’19]
- hQuick: distributed quicksort
- our merge sort: MS-simple (no LCP-comp), MS (LCP-comp)
- our prefix doubling merge sort: PDMS-Golomb, PDMS

# $D/N$-Generator($n=p\cdot 500$K, $\ell=500$, $D/N=?$)



# Conclusion

## Summary

- two new communication-efficient string sorting algorithms:
  - distributed string merge sort (MS)
  - distributed prefix-doubling string merge sort (PDMS)
- theory and experimental evaluation
- different strategies best for low and high $D/N$-ratios
- Source code and recording of talk: https://panthema.net/2020/0518-distributed-string-sorting

## Future Work

- improve balancing by considering strings and characters
- can one show lower bounds?

Questions via email to bingmann@kit.edu