Concurrency Effects Over Variable-size Identifiers in Distributed Collaborative Editing

ConcoRDanT

Brice Nédelec
Pascal Molli Achour Mostefaoui Emmanuel Desmontils

LINA, 2 rue de la Houssinière
BP02208, 44322 Nantes Cedex 03
first.last@univ-nantes.fr

UNIVERSITÉ DE NANTES

lina LABORATOIRE D'INFORMATIQUE DE NANTES ATLANTIQUE

LSEQ, an allocation strategy for variable-size CRDTs

All is about the insert operation

- insert(id_p, element, id_q) $\Rightarrow$ **alloc**(id_p, id_q) $\Rightarrow$ $id_{element}$ such as $id_p < id_{element} < id_q$

Allocation strategy before

- *linear*
- *editing behaviour dependant* $\Rightarrow$ design for end-editing

0 **1** 9

A

LSEQ allocation strategy

- lowered the space complexity : linear $\Rightarrow$ polylogarithmic
- any editing behaviour
  - $\Rightarrow$ variable-size CRDTs safe
  - $\Rightarrow$ avoids additional protocol
  - $\Rightarrow$ ready to use in editors ?

LSEQ, an allocation strategy for variable-size CRDTs

All is about the insert operation

- insert(id_p, element, id_q) $\Rightarrow$ **alloc**(id_p, id_q) $\Rightarrow$ $id_{element}$ such as $id_p < id_{element} < id_q$

Allocation strategy before

- *linear*
- *editing behaviour dependant* $\Rightarrow$ design for end-editing

0 1 **4** 9

A **N**

LSEQ allocation strategy

- lowered the space complexity : linear $\Rightarrow$ polylogarithmic
- any editing behaviour
  - $\Rightarrow$ variable-size CRDTs safe
  - $\Rightarrow$ avoids additional protocol
  - $\Rightarrow$ ready to use in editors ?

LSEQ, an allocation strategy for variable-size CRDTs

All is about the insert operation

- insert(id_p, element, id_q) $\Rightarrow$ **alloc**(id_p, id_q) $\Rightarrow$ $id_{element}$ such as $id_p < id_{element} < id_q$

Allocation strategy before

- *linear*
- *editing behaviour dependant* $\Rightarrow$ design for end-editing

0		1	4	**7**	**8**		9
		A	N	**G**	**E**	**S**	

LSEQ allocation strategy

- lowered the space complexity : linear $\Rightarrow$ polylogarithmic
- any editing behaviour
  - $\Rightarrow$ variable-size CRDTs safe
  - $\Rightarrow$ avoids additional protocol
  - $\Rightarrow$ ready to use in editors ?

LSEQ, an allocation strategy for variable-size CRDTs

All is about the insert operation

- insert(id_p, element, id_q) $\Rightarrow$ **alloc**(id_p, id_q) $\Rightarrow$ $id_{element}$ such as $id_p < id_{element} < id_q$

Allocation strategy before

- *linear*
- *editing behaviour dependant* $\Rightarrow$ design for end-editing

0		1	4	7	8	**8.1**	9
		A	N	G	E	S	

LSEQ allocation strategy

- lowered the space complexity : linear $\Rightarrow$ polylogarithmic
- any editing behaviour
  - $\Rightarrow$ variable-size CRDTs safe
  - $\Rightarrow$ avoids additional protocol
  - $\Rightarrow$ ready to use in editors ?

LSEQ, an allocation strategy for variable-size CRDTs

All is about the insert operation

- insert(id_p, element, id_q) $\Rightarrow$ **alloc**(id_p, id_q) $\Rightarrow$ $id_{element}$ such as $id_p < id_{element} < id_q$

Allocation strategy before

- *linear*
- *editing behaviour dependant* $\Rightarrow$ design for end-editing

0	**0.1**	1	4	7	8	8.1	9
	H	A	N	G	E	S	

LSEQ allocation strategy

- lowered the space complexity : linear $\Rightarrow$ polylogarithmic
- any editing behaviour
  - $\Rightarrow$ variable-size CRDTs safe
  - $\Rightarrow$ avoids additional protocol
  - $\Rightarrow$ ready to use in editors ?

LSEQ, an allocation strategy for variable-size CRDTs

All is about the insert operation

- insert(id_p, element, id_q) $\Rightarrow$ **alloc**(id_p, id_q) $\Rightarrow$ $id_{element}$ such as $id_p < id_{element} < id_q$

Allocation strategy before

- *linear*
- *editing behaviour dependant* $\Rightarrow$ design for end-editing

0	**0.0.1**	0.1	1	4	7	8	8.1	9
	C	H	A	N	G	E	S	

LSEQ allocation strategy

- lowered the space complexity : linear $\Rightarrow$ polylogarithmic
- any editing behaviour
  - $\Rightarrow$ variable-size CRDTs safe
  - $\Rightarrow$ avoids additional protocol
  - $\Rightarrow$ ready to use in editors ?

LSEQ, an allocation strategy for variable-size CRDTs

All is about the insert operation

- insert(id_p, element, id_q) $\Rightarrow$ **alloc**(id_p, id_q) $\Rightarrow$ $id_{element}$ such as $id_p < id_{element} < id_q$

Allocation strategy before

- *linear*
- *editing behaviour dependant* $\Rightarrow$ design for end-editing

0	**etc. . .**	0.0.1	0.1	1	4	7	8	8.1	9
	D	C	H	A	N	G	E	S	

LSEQ allocation strategy

- lowered the space complexity : linear $\Rightarrow$ polylogarithmic
- any editing behaviour
  - $\Rightarrow$ variable-size CRDTs safe
  - $\Rightarrow$ avoids additional protocol
  - $\Rightarrow$ ready to use in editors ?

Not yet...

left open in LSEQ perspectives : the **concurrency effects**

- how the LSEQ **alloc** function behaves when
  1. varying number of users
  2. varying latency

	#insert operations				
	10	**100**	**200**	**500**	**1000**
1 user (bit/id)	6.5	26.8	32.7	56.0	64.2
10 users (bit/id)	9.5	125.8	377.0	1962.1	5468.0

- expectation : sub-linear behaviour
- reality : quadratic growth

$\Rightarrow$ only partially improves variable-size identifiers

Objective

- Solve the limitation of LSEQ in context involving concurrency
  - **Extends** sub-linear upper-bound of single user to **multiple users**
  - **Study** the effect of **latency** over the size of identifiers

  $\Rightarrow$ No costly additional protocol
  $\Rightarrow$ I can build a Distributed Collaborative Editor based on variable-size CRDTs
    - Decentralized
    - Simple algorithms
    - Tiny metadata

Proposal : **h-LSEQ**

Similar to LSEQ

- exponential tree model
- multiple sub allocation strategies
  - designed for end-editing
  - designed for front-editing

Different

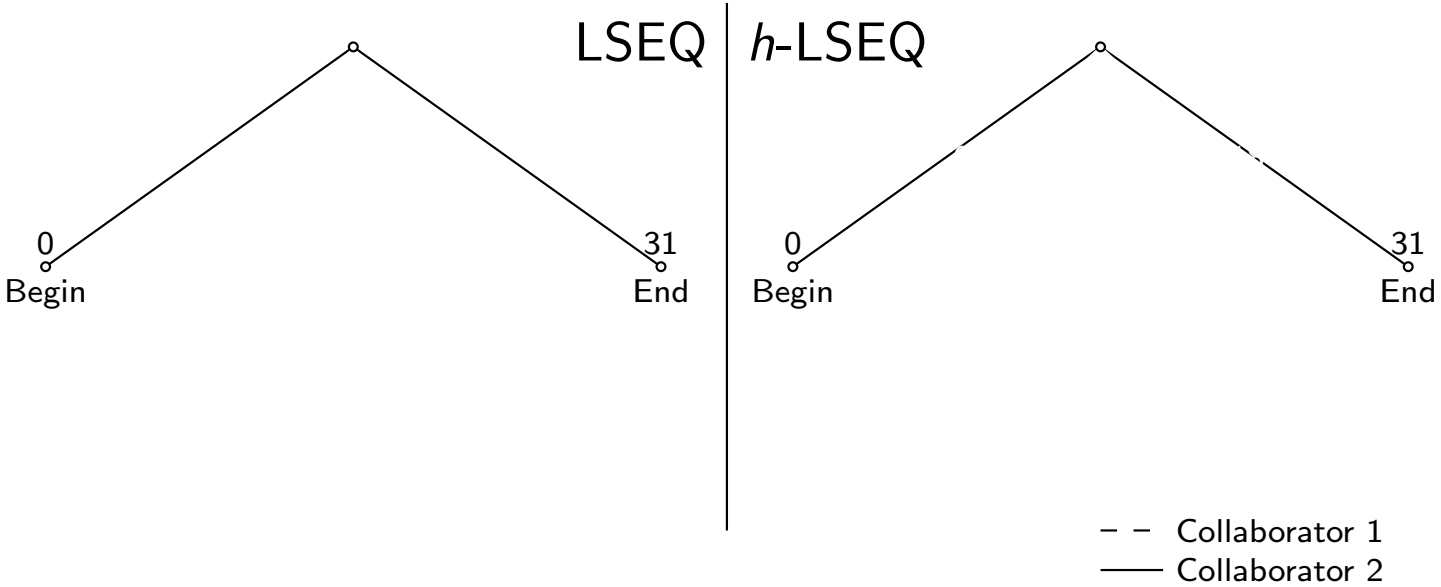
- strategy choice

	local	different
LSEQ	random	different
h-LSEQ	random	**similar**

Intuition

The shared hash function provide an *a priori* **agreement** over participants on which strategy to employ. This agreement **avoids** the possibility of **antagonist choices** which would have led to a bad global allocation of identifiers.

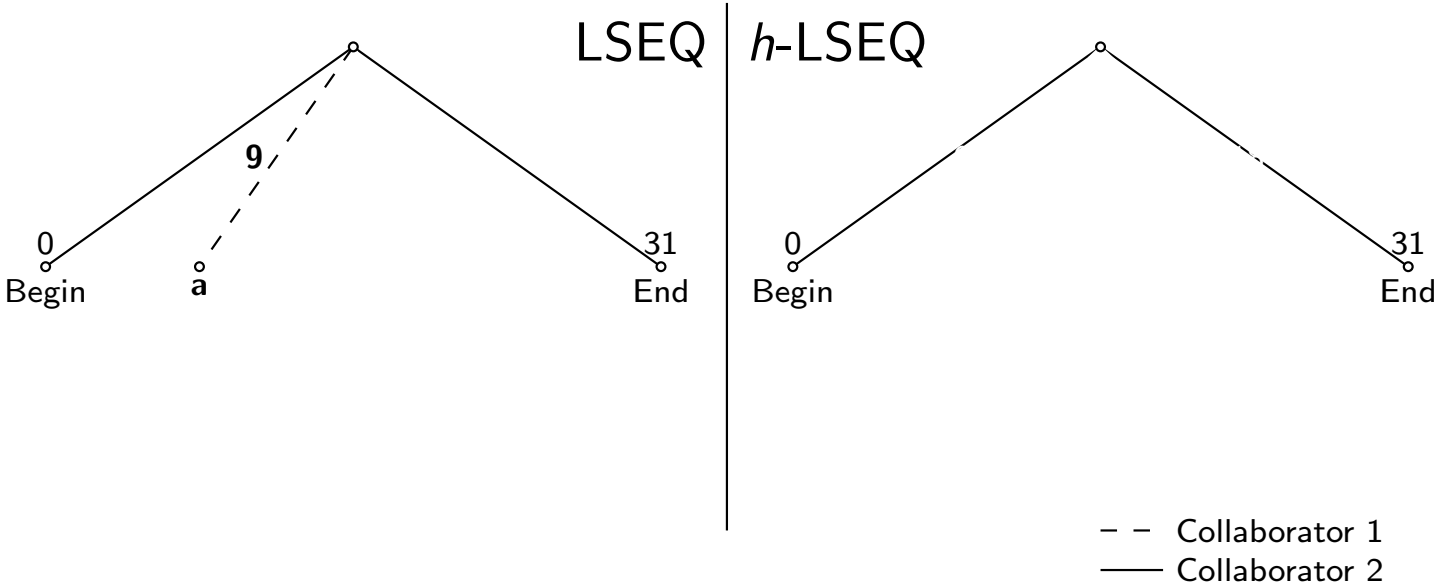
LSEQ vs *h*-LSEQ : example



1. user1 : insert(B,a,E)
   - ⇒ lvl 1 → End-editing
     - ⇒ [9]
2. user2 : insert(a,b,E)
   - ⇒ lvl 1 → Front-editing
     - ⇒ [30]
3. repeat 1 x : depth increases
4. repeat n x : quadratic growth

1. user1 : insert(B,a,E)
   - ⇒ lvl1 → End-editing
     - ⇒ [9]
2. user2 : insert(a,b,E)
   - ⇒ lvl1 → End-editing
     - ⇒ [10]
3. repeat 1 x : stay lvl 1
4. repeat n x : sub-linear behaviour

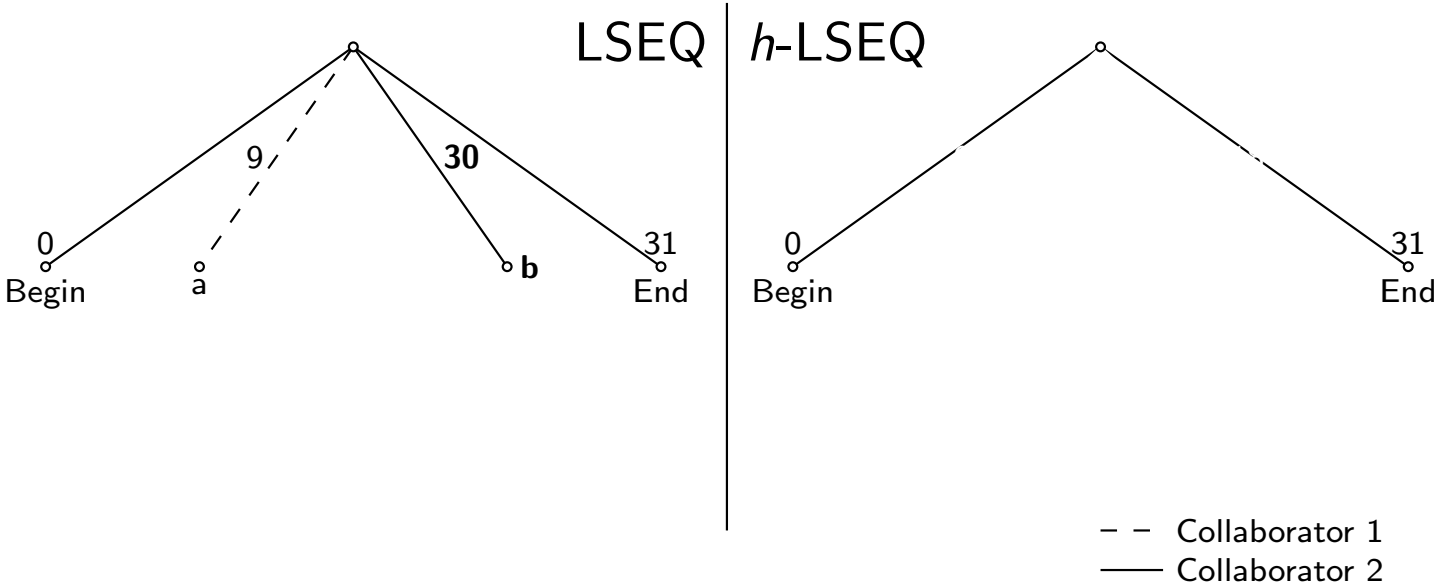
LSEQ vs *h*-LSEQ : example



1. user1 : insert(B,a,E)
   - ⇒ lvl 1 → End-editing
     - ⇒ [9]
2. user2 : insert(a,b,E)
   - ⇒ lvl 1 → Front-editing
     - ⇒ [30]
3. repeat 1 x : depth increases
4. repeat n x : quadratic growth

1. user1 : insert(B,a,E)
   - ⇒ lvl1 → End-editing
     - ⇒ [9]
2. user2 : insert(a,b,E)
   - ⇒ lvl1 → End-editing
     - ⇒ [10]
3. repeat 1 x : stay lvl 1
4. repeat n x : sub-linear behaviour

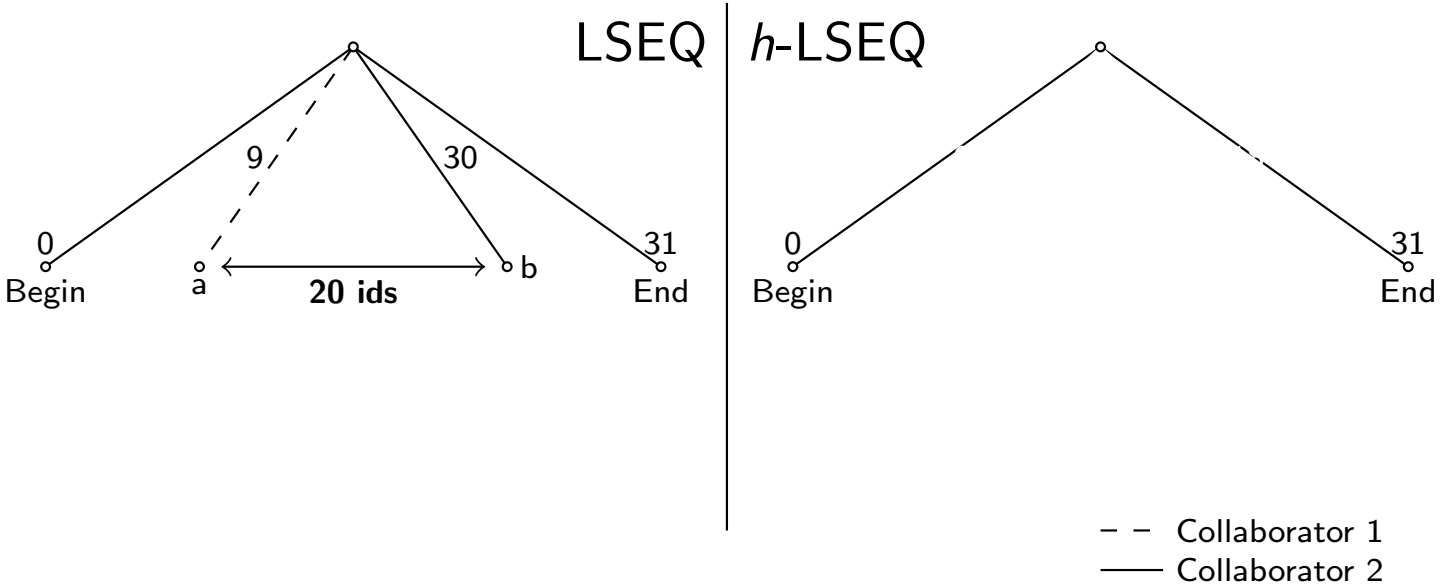
LSEQ vs *h*-LSEQ : example



1. user1 : insert(B,a,E)
   - ⇒ lvl 1 → End-editing
     - ⇒ [9]
2. user2 : insert(a,b,E)
   - ⇒ lvl 1 → Front-editing
     - ⇒ [30]
3. repeat 1 x : depth increases
4. repeat n x : quadratic growth

1. user1 : insert(B,a,E)
   - ⇒ lvl1 → End-editing
     - ⇒ [9]
2. user2 : insert(a,b,E)
   - ⇒ lvl1 → End-editing
     - ⇒ [10]
3. repeat 1 x : stay lvl 1
4. repeat n x : sub-linear behaviour

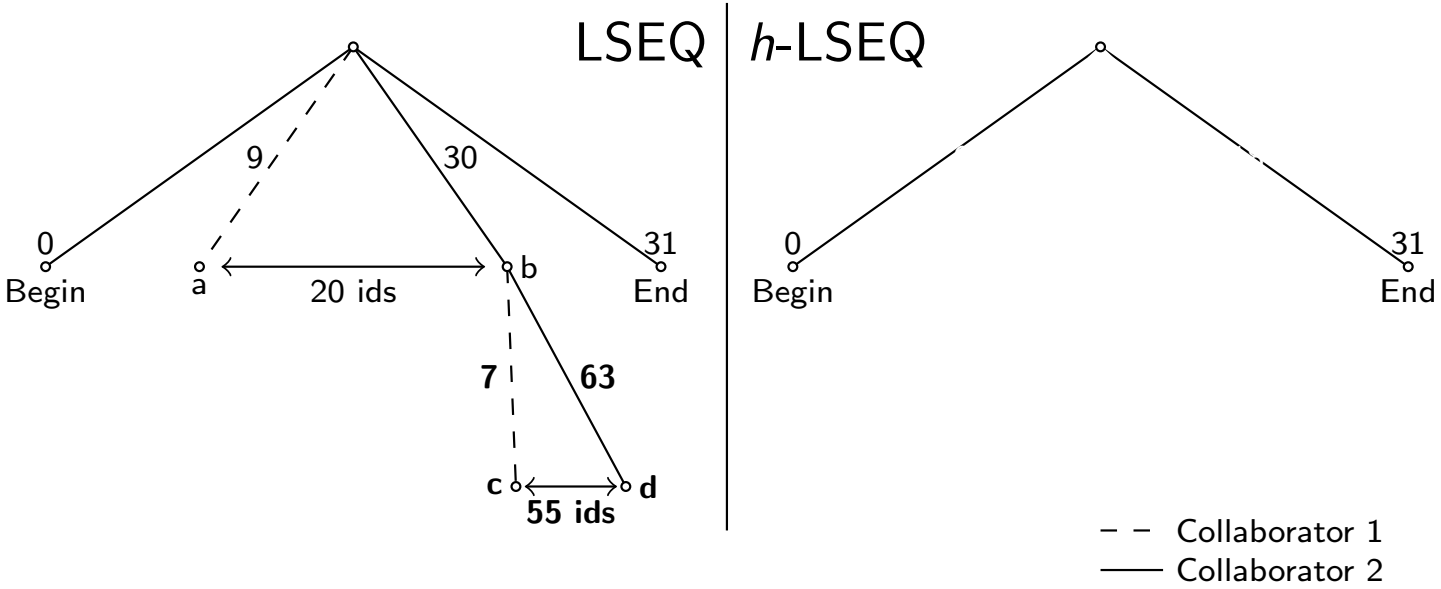
LSEQ vs *h*-LSEQ : example



1. user1 : insert(B,a,E)
   - ⇒ lvl 1 → End-editing
     - ⇒ [9]
2. user2 : insert(a,b,E)
   - ⇒ lvl 1 → Front-editing
     - ⇒ [30]
3. repeat 1 x : depth increases
4. repeat n x : quadratic growth

1. user1 : insert(B,a,E)
   - ⇒ lvl1 → End-editing
     - ⇒ [9]
2. user2 : insert(a,b,E)
   - ⇒ lvl1 → End-editing
     - ⇒ [10]
3. repeat 1 x : stay lvl 1
4. repeat n x : sub-linear behaviour

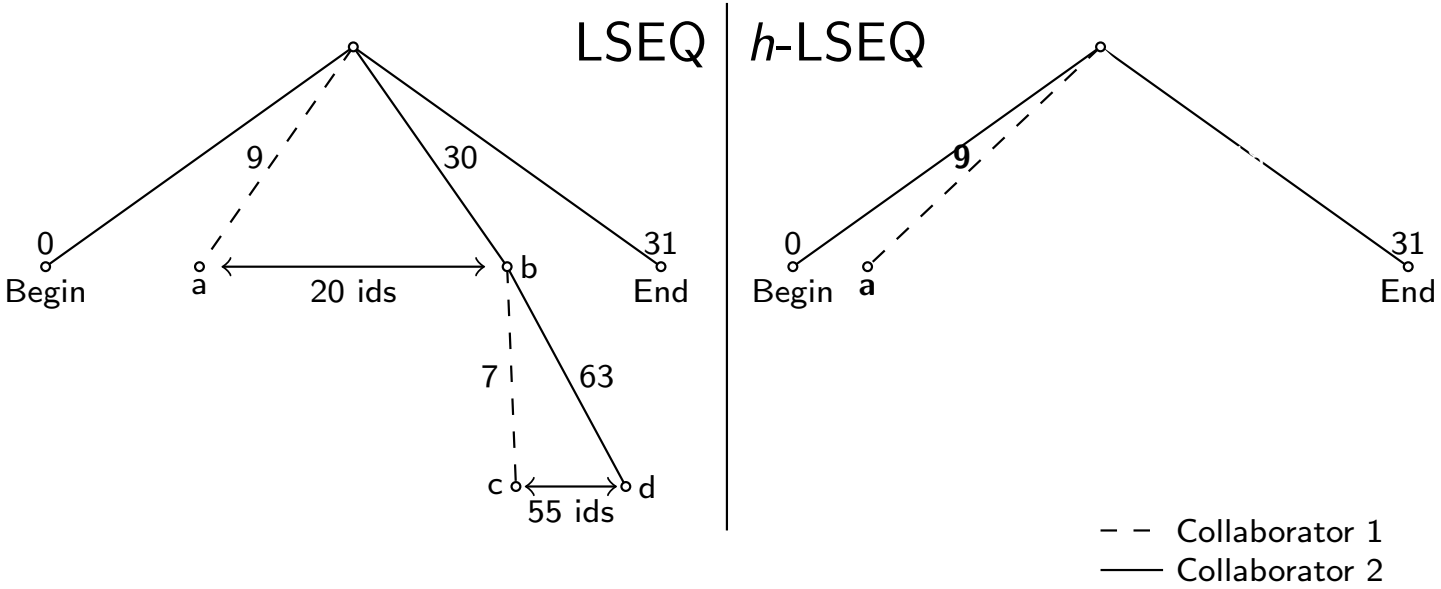
LSEQ vs *h*-LSEQ : example



1. user1 : insert(B,a,E)
   - ⇒ lvl 1 → End-editing
     - ⇒ [9]
2. user2 : insert(a,b,E)
   - ⇒ lvl 1 → Front-editing
     - ⇒ [30]
3. repeat 1 x : depth increases
4. repeat n x : quadratic growth

1. user1 : insert(B,a,E)
   - ⇒ lvl1 → End-editing
     - ⇒ [9]
2. user2 : insert(a,b,E)
   - ⇒ lvl1 → End-editing
     - ⇒ [10]
3. repeat 1 x : stay lvl 1
4. repeat n x : sub-linear behaviour

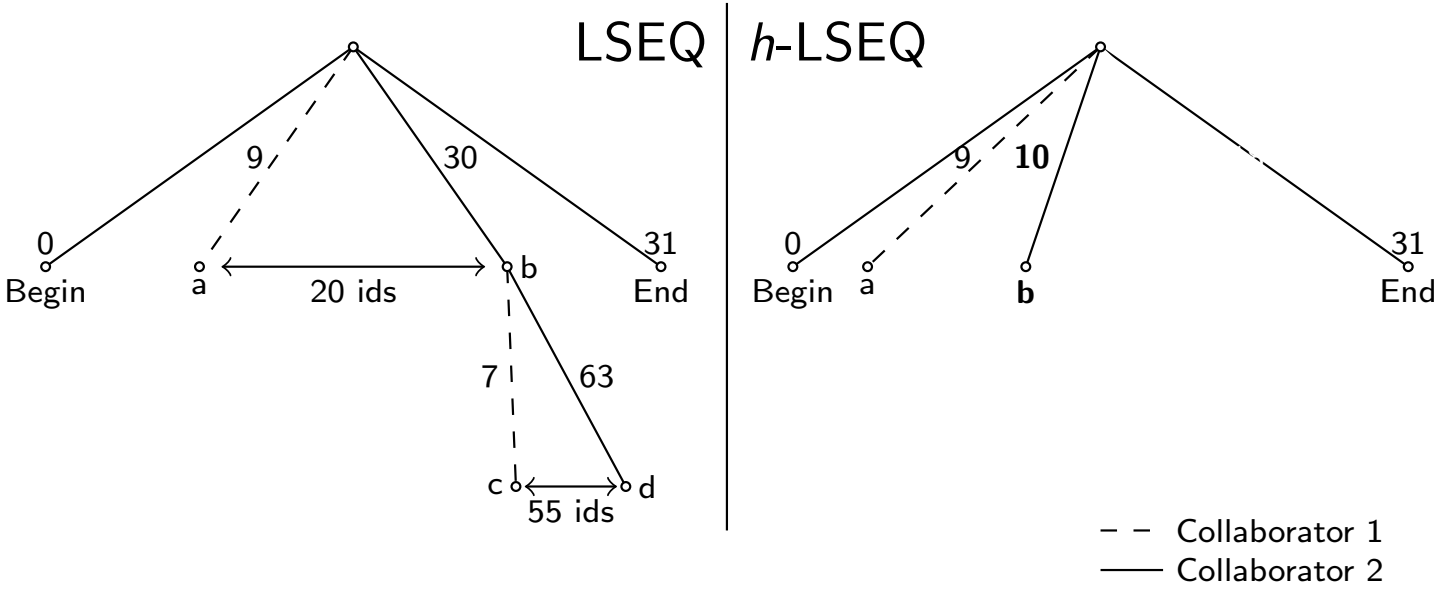
LSEQ vs *h*-LSEQ : example



1. user1 : insert(B,a,E)
   - ⇒ lvl 1 → End-editing
     - ⇒ [9]
2. user2 : insert(a,b,E)
   - ⇒ lvl 1 → Front-editing
     - ⇒ [30]
3. repeat 1 x : depth increases
4. repeat n x : quadratic growth

1. user1 : insert(B,a,E)
   - ⇒ lvl1 → End-editing
     - ⇒ [9]
2. user2 : insert(a,b,E)
   - ⇒ lvl1 → End-editing
     - ⇒ [10]
3. repeat 1 x : stay lvl 1
4. repeat n x : sub-linear behaviour

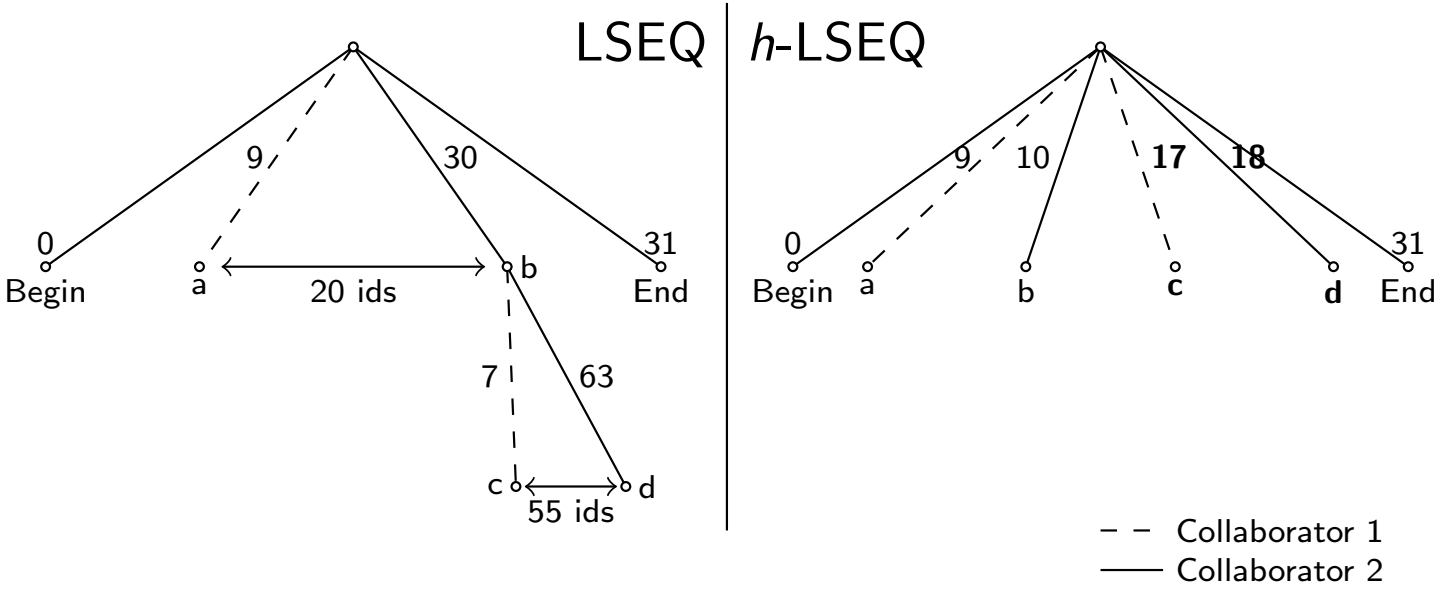
LSEQ vs *h*-LSEQ : example



1. user1 : insert(B,a,E)
   - ⇒ lvl 1 → End-editing
     - ⇒ [9]
2. user2 : insert(a,b,E)
   - ⇒ lvl 1 → Front-editing
     - ⇒ [30]
3. repeat 1 x : depth increases
4. repeat n x : quadratic growth

1. user1 : insert(B,a,E)
   - ⇒ lvl1 → End-editing
     - ⇒ [9]
2. user2 : insert(a,b,E)
   - ⇒ lvl1 → End-editing
     - ⇒ [10]
3. repeat 1 x : stay lvl 1
4. repeat n x : sub-linear behaviour

LSEQ vs *h*-LSEQ : example



1. user1 : insert(B,a,E)
   - ⇒ lvl 1 → End-editing
     - ⇒ [9]
2. user2 : insert(a,b,E)
   - ⇒ lvl 1 → Front-editing
     - ⇒ [30]
3. repeat 1 x : depth increases
4. repeat n x : quadratic growth

1. user1 : insert(B,a,E)
   - ⇒ lvl1 → End-editing
     - ⇒ [9]
2. user2 : insert(a,b,E)
   - ⇒ lvl1 → End-editing
     - ⇒ [10]
3. repeat 1 x : stay lvl 1
4. repeat n x : sub-linear behaviour

Experiments

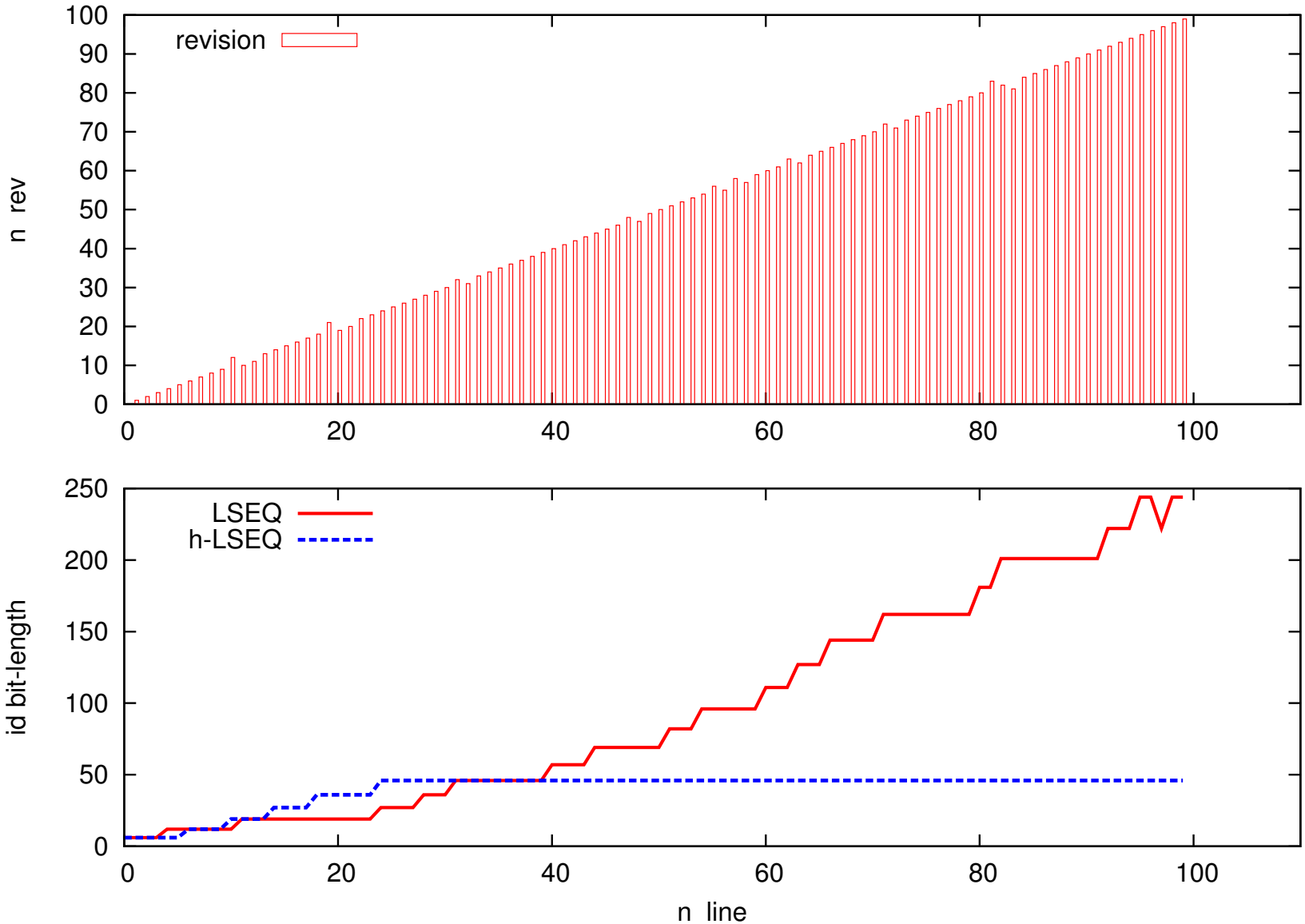
1. Evaluation of LSEQ and *h*-LSEQ on a collaboration involving **multiple users**
   - $\Rightarrow$ Synthetic documents (100 lines)
   - $\Rightarrow$ 10 users (10 op/each)
   - $\Rightarrow$ Editing behaviour : end editing
   - $\Rightarrow$ Instant delivery of messages ($\approx$ LAN)

   Expect :
   - a. LSEQ $\Rightarrow$ quick growth of identifiers
   - b. *h*-LSEQ $\Rightarrow$ quick stabilization $\Rightarrow$ sub-linear upper-bound

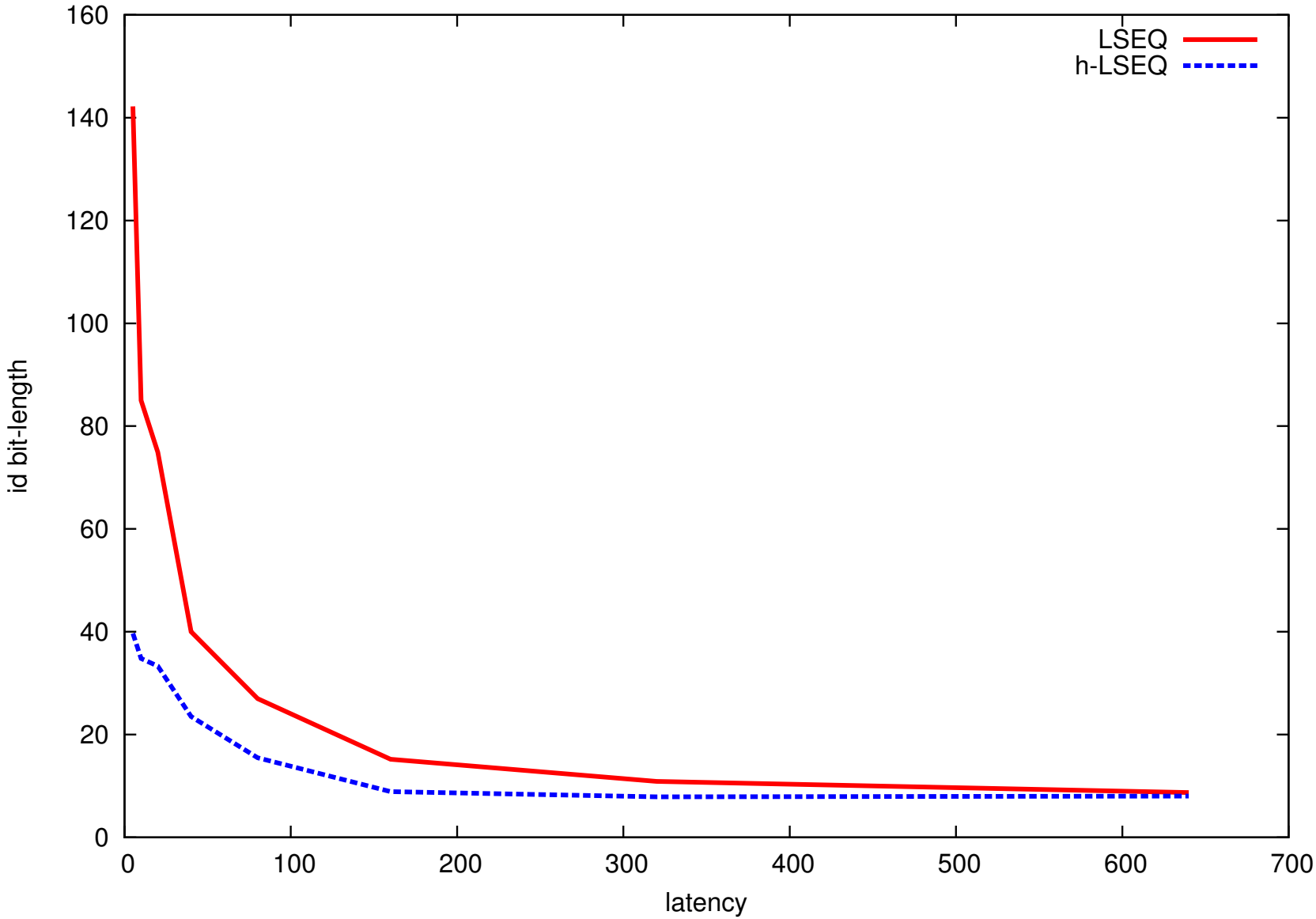
2. Evaluation of LSEQ and *h*-LSEQ with **varying latency**
   - $\Rightarrow$ Synthetic documents (100 lines)
   - $\Rightarrow$ 10 users (10 op/each)
   - $\Rightarrow$ Editing behaviour : end editing

   Expect :
   - a. no latency : worst-case
   - b. latency $\nearrow$ then avg(id.size) $\searrow$

h-LSEQ scales in term of users



Latency does not badly impact variable-size CRDTs



Synthesis : experiments

1. *h*-LSEQ $\Rightarrow$ hash-based choice strategy $\Rightarrow$ global agreement on employed strategies
   - $\Rightarrow$ No additionnal cost (only a shared seed within the document)
   - $\Rightarrow$ Generalize the space complexity from single user to multiple users
2. Latency :
   - No bad impact on size of identifiers
   - No latency $\Rightarrow$ upper-bound on the size of identifiers

$\Rightarrow$ *h*-LSEQ supports concurrency

Conclusion

h-LSEQ handles concurrency

$\Rightarrow$ limitation solved

With *h*-LSEQ :

- Distributed Collaborative Editors
  - Decentralized
  - Very simple algorithms
  - tiny metadata

CRDT-based distributed collaborative editors using h-LSEQ constitutes a good alternative to trending editors such as Google Docs, Etherpad. . . with better scalability.

Future works

1. Develop the Distributed Collaborative Editor
2. Proof
   - n operations : uniform distribution $\Rightarrow O((log\ log\ n)^2)$
   - n operations : monotononic $\Rightarrow O((log\ n)^2)$
   - n operations : worst-case $\Rightarrow O(n^2)$
3. Proof : worst-case happens with a negligible probability
4. Causality tracking (still an issue in distributed systems with churn)
   - does not scale in term of user
   - or does not provide exact causality

   $\Rightarrow$ CRDTs for sequences require causality...

Thank you !