

Problem Statement

Concurrency control in a
distributed collaborative text
editor with **multi-character**
insertion and deletion.

Problem

Concurrency
Control

Multiple concurrent
operations must always
produce a deterministic
output

Distributed

Operations must be
tolerant to race conditions
without a global view of the
network

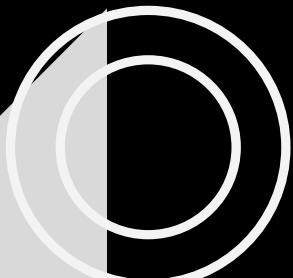
Multi-character
operations

Batch operations together
to optimise for
performance

Algorithms

Operational
Transformation

Conflict-Free
Replicated
Data Types



Operational Transformation

A method to resolve conflicts in a concurrent system by **modifying the operations** that hosts in the system send and receive.

Downside:

- Requires a global view of the entire network
- No centralized server or manager




Conflict-Free Replicated Data Types (CRDT)

A data structure that can be replicated and updated **independently and concurrently**.

Sequence CRDTs

An ordered set with the properties of a CRDT.






Sequence CRDT

A sequence CRDT is an ordered set that has all the properties of a CRDT.

It is possible to use a sequence CRDT as the data structure of a real-time collaborative text editor.



LSEQ

The main source of entropy in a text document is the randomness involved in text editing.

- Writers usually edit, delete, and rearrange text more often than they add characters to the document.

Therefore, it is difficult to predict patterns that optimize our sequence CRDT for text editing.

- How should we arrange the nodes in our tree to minimize its depth?
- Deeper trees require more memory to store each node's position, which acts as the GUID for each element.

LSEQ is a sequence CRDT allocation strategy that aims to amortise the depth of a tree by exponentially increasing the number of child nodes and assigning a random “direction” to each node.

Batch Operations

Performance gains in
time

Sequentially generating k GUIDs: $O(k \log n)$

Batch generating k GUIDs: $O(k)$

Sequentially deleting k elements from the array:
 $O(k^2)$

Batch deleting k elements by swapping: $O(k)$

Performance gains in
memory

Adjacent GUIDs maximise the number of children per node.

Therefore, the depth of the tree is minimized, which minimizes memory usage.

Performance

Base Size	Average Depth
2	5.25
4	4.49
8	3.73
16	3.59
32	3.42
64	3.31
128	3.21

Moving forward

2D structure
for text

Recursive
serialization
of GUIDs

Extend our
network with
WebRTC

2D structure for text

	Search	Insert / Delete
Linear Array	$O(\log N)$	$O(N)$
Two Dimensional Array	$O(\log L + \log C)$	$O(C)$

N = number of elements

L = number of lines

C = average number of elements per line

$$N = L \times C$$

Recursive serialization of GUIDs

Currently, all GUIDs are serialized and stored when serializing a list of GUIDs.

`[{1,2,3}, {1,2,4}, {1,2,6}, {1,3}]`

However, this does not take advantage of the tree structure of the GUIDs.

In the future, we would like to explore more efficient ways of serializing a list of GUIDs by representing the tree structure.

Extend our network with WebRTC

As we built our Data Structure and Algorithm in Java, we are unable to use **WebRTC**, as there are no supporting libraries.

- The WebRTC protocol allows browsers to establish direct connections with each other.

We believe will benefit in establishing connections directly between peers.

