YJS Deep Dive and Analysis

Moin Ahmed Qidwai

2021-09-26

0.1 Introduction

Near-real time (NRT) collaboration is the goal of many software applications, from collaborative text-editors like Google Docs to modelling applications like Autodesk Maya. In reality most applications could benefit from allowing users to collaborate effectively regardless of the problem domain that the application targets. As such the reason that so few of the mainstream software supports this functionality is generally the result of complexity accompanied with solutions to this problem. While there are a few different methodologies for supporting NRT collaboration, in this paper we shall investigate YJS, a popular library implementing the YATA approach.

0.2 Collaboration Objectives

In order for a solution to be considered for NRT collaboration, it must be able to satisfy certain conditions or objectives.

0.2.1 Eventual Convergence

Eventual convergence dictates that if two collaborators receive the same set of operations in any order, the end result of those operations must be the same for both the collaborators.
That is, If we have a algorithm A for merging operations into a list and two sets of operations S_1 and S_2 that observe the below relation.

$$\forall o : o \in S_1 \leftrightarrow o \in S_2 \tag{1}$$

Then the following must hold true, where $\mapsto$ represents an input symbol.

$$S_1 \mapsto A \equiv S_2 \mapsto A \tag{2}$$

0.2.2 Intention Preservation

The idea behind preservation of intention is simple. Any solution that aims to provide for NRT collaboration must ensure the result is in accordance with the intention of all collaborators.

0.2.3 Interleaving

Interleaving occurs if two or more collaborators insert multiple characters at the same index, upon integrating their insertions the result may have their inputs mixed.
Example: Collaborators C_1 and C_2 add "vious" and "cious" to "pre". If interleaving occurs the result may be "prevcioiouuss". A program that allows for NRT collaboration must ensure interleaving cannot occur.

0.3 YATA

The YATA (Yet Another Transformation approach) is the core specification underlying YJS. This specification consists of two main components, a doubly linked list and a set of rules that all operations must observe.

0.3.1 Data Representation

The doubly linked list representation used by YATA is in contrast to other algorithms. Another popular algorithm is the RGA, which utilizes a unidirectional linked list. The doubly linked list allows YATA to avoid interleaving at the start of the document or in prepend operations. As such it can cater for a wider range of operations and use cases than RGA by default. Though due to storing a pointer to the successor the data representation in YATA requires more memory.

$$Block_i = (id_i, origin_{left}, origin_{right}, deleted_i, value_i) \tag{3}$$

Equation 3 represents a single element in the linked list of YATA. The origins represent the pointers to predecessor and successor elements.

$$id_i = (replica_i, counter_i) \tag{4}$$

Equation 4 represents the identifier for a single block in the linked list. It consists of the Replica ID (or user id) and the operation counter.

The above representation ensures each block has a unique identifier and a total order.

0.3.2 Operations

The YATA specification only outlines two types of operations: insertion and deletion. A combination of these operations can also lead to many others, for example the update operation.

Insertion

$$Operation_k = (id_k, origin_k, left_k, right_k, deleted_k, value_k) \tag{5}$$

Equation 5 represents the insertion operation with counter (k). The **origin** represents the predecessor for the block at the time of creation. The **left** and **right** represent the predecessor and successor respectively after the operation has been merged into the linked list. The **deleted** flag indicates if the block representing the operation has been marked for deletion. The **value** is the actual content that is to be inserted.

Deletion

The deletion operation is simply represented by setting the deleted flag of the insertion operation to **true**.

$$Operation_k = (id_k, origin_k, left_k, right_k, true, value_k) \tag{6}$$

Operation Ordering

Every operation block has a total order in the list defined by the natural predecessor relation $<$.

$$O_1 < O_2 \leftrightarrow O_1 is\ a\ predecessor\ of\ O_2 \tag{7}$$

$$O_1 \leq O_2 \leftrightarrow O_1 < O_2 \vee O_1 \equiv O_2 \tag{8}$$

Given the above predecessor relation and insert operation we can represent an insertion between two operations $O_i\ and\ O_j$ as shown below.

$$Operation_{new} = (id_{new}, O_i, O_i, O_j, false, value_{new}) \tag{9}$$

In equation 9 the following relation must hold $O_i < Operation_{new} < O_j$.

As one may notice the origin and the left operation are the same in the

above equation as this represents the operation at the time of it's creations. The origin for the operation is set at the time of the operation creation and does not change thereafter. The left pointer may change during the merge process of operations created by different replicas, if there are conflicts.

0.3.3 Rules of Conflict Resolution

As mentioned earlier in this paper YATA consists of certain rules that must be observed by operations specially in cases of conflicts. These rules are the cornerstone of the YATA approach as they ensure **eventual convergence** and **intention preservation**.

Conflicting insertions

Insertion operations (O_a, O_b, ...) are in conflict if all of them are to be inserted between O_i and O_j. In the above example, if $Operation_{new}$ is to be integrated in the list of operations $L = [O_i, c_a, c_b, c_c ... O_j]$, then $Operation_{new}$ is in conflict with $[c_a, c_b, c_c, ...]$. The rules resolve these conflicts by calculating the index k for the new insertion. If the rules are observed by all collaborators then each of them calculates the same index. Each rule can be illustrated as a predecessor relation $<_r$. As such if we observe these rules for $Operation_{new}$ then we integrate it between c_i and c_j where $\forall r : c_i <_r Operation_{new} <_r c_j$.

Rule One

The first rule dictates that for two conflicting insertions I_a and I_b that have different origins O_a and O_b respectively, the connection between I_a and O_a must not be intersected by the connection between I_b and O_b, or vice versa. The only instances where the above holds true is illustrated by the below ordered sets (and their opposites, which one can get by swapping the indexes a and b).

$$[O_a, O_b, I_b, I_a] \tag{10}$$

$$[O_a, I_a, O_b, I_b] \tag{11}$$

Set 10 represents the case where a operation and it's origin are inserted in between of the other operation and it's origin. Set 11 represents the case where a operation and it's origin are inserted after the other operation and it's origin.

Rule one then can be succinctly illustrated by the below equation.

$$O_a <_{r1} O_b \leftrightarrow O_a < Origin_b \vee Origin_b \leq Origin_a \tag{12}$$

Rule Two

Rule two is the standard rule of transitivity and can be illustrated by the below equation.

$$O_a <_{r2} O_b \leftrightarrow \forall O : O_b <_{r2} O \rightarrow O_a \leq O \tag{13}$$

Rule Three

Rule three dictates that if two conflicting insertions have the same origin, then the insertion with the smaller creator ID is to the left. It can be represented by the below equation.

$$O_a <_{r3} O_b \leftrightarrow Origin_a \equiv Origin_b \rightarrow Creator_a < Creator_b \tag{14}$$

Total Order Function

If we combine all the three rules by conjunction and utilize them for insertions we get a total order on the insertion operations. This ensures both **eventual convergence** and **intention preservation**.

0.4 YJS

YJS is an implementation of the YATA specification, though it does couple it with delta-state based operations. In other words it only passes the specific operations that changed per integration, as opposed to the full document as per the YATA specification. This ensures the size of the messages remains small and hence the burden on the network resources is minimized.
The main disadvantage of YJS along with YATA is that when each character is represented as an operation as opposed to a single character, the overall document takes greater space. Though this is compensated with generally lower time complexity and the small size of the propagated messages.

0.5 Conclusion

The YATA approach resolves many of the short-comings of other NRT collaboration approaches. It can be utilized to represent a multitude of data types and hence can cater for a wide range of applications. YJS is the most popular implementation of YATA, though it adds many optimizations beyond the core. It also has significant advantages over operational transformation based techniques, which we will look at next.