АВТОМАТИЧЕСКОЕ РАЗРЕШЕНИЕ КОНФЛИКТОВ ПРИ СИНХРОНИЗАЦИИ РЕПЛИК В РАСПРЕДЕЛЁННОЙ СИСТЕМЕ

Сафин К. В.
Кафедра инженерной психологии и эргономики,
Белорусский государственный университет информатики и радиоэлектороники
Минск, Республика Беларусь
E-mail: safinkaren7@gmail.com

В данной работе излагаются особенности бесконфликтных реплицированных типов данных, а также описаны технологии операционных преобразований для разрешения конфликтов в приложениях совместного редактирования

Введение

Совместная работа с такими приложениями как календарь, сервис заметок типа Evernote или сервис совместного редактирования документов, таких как Google Doc, сложна с технической точки зрения, потому что несколько людей могут вносить различные изменения в одни и те же данные в практически одинаковые моменты времени. Так как доставка данных через интернет не осуществляется мгновенно, то для имитации мгновенного отклика в каждый момент времени клиент работает с локальной версией (репликой) редактируемого документа, которая может отличаться от версий других участников. Основная проблема в этом случае – обеспечить консистентность локальных версий.

Существует два основных подхода к решению настоящей проблемы, выбор которых зависит от особенностей реплицируемых данных. *Операционные трансформации* - подход, применяемый обычно при работе с текстом и состоящий в безконфликтном преобразовании слиянии данных. Имеет более высокие требования к серверу, более сложные и менее стабильные алгоритмы [3]. Второй подход состоит в использовании *бесконфликтных реплицируемых типов данных*, особенности которых рассмотрены ниже.

I. Операционные трансформации

Рассмотрим один из алгоритмов операционного трансформирования, применяемый при совместном редактировании текста [5].

Допустим, что все изменения отправляются на сервер в виде набора команд:

$$(\ell_1 \to \ell_2)[c_1, c_2, c_3, ...].$$

где:

- ℓ_1: длина документа до редактирования;
- ℓ_2: длина документа после редактирования.
- $c_1, c_2, c_3, ...$ — параметры описания документа после редактирования.

Если c_i – число (или диапазон чисел), то это порядковый номер (номера) символа исходного документа (далее индексы), который останется после редактирования, а если c_i – символ (или строка), то это вставка новых символов.

Пример:

В исходную строку “” приходят изменения $(0 \to 2)[“Hi”]$, получается “Hi”. После этого приходят изменения $(2 \to 5)[1, “ello”]$, получаем “$Hello$”. Итоговый документ формируется как последовательность таких изменений, применённых по порядку к исходному состоянию общего документа (здесь и далее обозначаемое как X).

Далее несколько реплик могут быть обьедены в общий документ путем так называемой операции *слияния* $m(A, B)$, где A и B это команды редактирования от двух разных реплик: $A = (n \to n_a)[\cdots]$ и $B = (n \to n_b)[\cdots]$ соответсвтенно. Операция слияния обязана обладать свойством коммутативности:

$$m(A, B) = m(B, A).$$

Заметим, что для клиента с изменениями A, получившему изменения B, нет смысла вычислять $m(A, B)$, так как $m(A, B)$ применяется к X, а у A текущее состояние $A(X)$. В этом случае, нам нужно вычислить A' и B', такие, что:

$$B'(A(X)) = A'(B(X)) = m(A, B)(X).$$

Определим функцию вычисления A' и B':

$$f(A, B)(A) = f(B, A)(B) = m(A, B) = m(B, A)$$

где $m(A, B) = m(B, A) = A(B') = B(A')$. Алгорим построения $f(A, B)$ состоит из следующих шагов [3]:

- вставки в команде редактирования A становятся индексами в $f(A, B)$;
- вставка в команде редактирования B становятся вставкой в $f(A, B)$;
- совпадающие индексы в A и B переносятся в $f(A, B)$.

Пример:

Дана строка "baseball". Две распределенные системы параллельно вносят свои поправки в свои реплики этой строки. Вычислим результат слияния реплик путем операциональных трансформаций.

$$X = (0 \to 8)[“baseball”];$$

$$A = (8 \to 5)[0-1, \text{“}si\text{”}, 7](A = \text{“}basil\text{”});$$
$$B = (8 \to 5)[0, \text{“}e\text{”}, 6, \text{“}ow\text{”}](B = \text{“}below\text{”}).$$

Вычисляем:

$$A' = f(B, A) = (5 \to 6)[0, 1, \text{“}si\text{”}, 3, 4];$$
$$B' = f(A, B) = (5 \to 6)[0, \text{“}e\text{”}, 2, 3, \text{“}ow\text{”}];$$
$$m(A, B) = m(B, A) = (8 \to 6)[0, \text{“}esiow\text{”}].$$
$$m(A, B)(X) = (8 \to 6)[0, \text{“}e\text{”}, 2, \text{“}iow\text{”}]$$

Таким образом, изначальная строка "baseball" путем совместного редактирования двух распределенных систем, и последующего слияния реплик "basil" и "below" превратилась в строку "besilow".

II. Бесконфликтные реплицируемые типы данных (БРТД)

БРТД - это особый вид данных обладающий строгой конечной консистентностью. Это значит, что эти данные обязаны обладать тремя свойствами [2]:

- идемпотентность: $a \vee a = a$;
- ассоциативность: $(a \vee b) \vee c = a \vee (b \vee c)$;
- коммутативность: $a \vee b = b \vee a$.

Из известных и описанных, стоит отметить следующие [1]:

- G-Counter: (grow-only) монотонно увеличивающийся счётчик;
- PN-Counter: (positive-negative) счётчик, который можно уменьшать;
- LWW-Register: (last-writer-wins) регистр с принципом "последняя запись приоритетнее";
- MV-Register: (multi-value) регистр с несколькими значениями;
- G-Set: множество элементов без удаления;
- 2P-Set: множество элементов с приоритетным удалением;
- PN-Set: множество, которое использует счётчик операций включения-удаления;
- LWW-Set: множество с приоритетом времени операции;
- OR-Set: (observed-remove) множество с идентификаторами.
- Ordered-List: упорядоченный список.
- Graphs: структуры, описывающие вершины и связи в графах, основанные на списках.

Рассмотрим правило манипуляции упорядоченным списком, что бы он считался списком, обладающим строгой конечной консистентностью.

Пусть исходное состояние общего списка X, это список символов “$HELO$”. Присвоим каждому элементу списка парный идентификатор. Первый элемент идентификатора это порядковый номер элемента а второй - идентификатор списка: “$H_{0x}\ E_{1x}\ L_{2x}\ O_{3x}$”. При создании реплики, список копируется без изменений.

Допустим, что все изменения отправляются на сервер в виде набора команд из следующего списка возможных:

- для инициализации списка: (“$char$”)$[id]$; где id - идентификатор первого символа.
- для вставки символа: (“$char$”)$[id_1, id_2]$; где id_1 - идентификатор элемента перед вставляемым элементом а id_2 - идентификатор вставляемого элемента. Новый идентификатор должен иметь порядковый номер следующий за последним.
- для удаления символа: $[id]$. где id - идентификатор удаляемого элемента.

Например строка "HO" заменяется на "HI" , путем выполнения следующих команд:

$$H_{0x}O_{1x} \to [1x] \to (I)[0x, 2x] = H_{0x}I_{2x}$$

Пример для реплик A и B:

1. $h_{0x}e_{1x}l_{2x}o_{3x} \to (l)[2x, 4a] \to h_{0x}e_{1x}l_{2x}l_{4x}o_{3x}$;
2. $h_{0x}e_{1x}l_{2x}o_{3x} \to (!)[3x, 4b] \to h_{0x}e_{1x}l_{2x}o_{3x}!_{4b}$.

Далее рассмотрим правило обьединения двух конфликтующих реплик A и B :

1. $a_{0x}b_{1x}c_{2x} \to (x)[0x, 3a] \to a_{0x}x_{3a}b_{1x}c_{2x}$;
2. $a_{0x}b_{1x}c_{2x} \to (p)[0x, 3b] \to a_{0x}p_{3b}b_{1x}c_{2x}$.

Требуется вычислить результат слияния $m(A, B)$. Операция слияния БРТД обладает свойством коммутативности как и в случае с операционными трансформациями.

$$m(A, B) = m(B, A).$$

Предлагается следующий алгоритм слияния [4]:

При слиянии реплики B в A используются те же команды модификации что и при $B(X)$ за исключением операции вставки. После этого проверяется что бы идентификатор элемента id_i, следующего за элементом id_1 не был больше чем id_2. В случае если он больше, повторяем проверку для id_{i+1}. В итоге, найдя такой id_{i+n}, где $id_{i+n} < id_2$, вставляем элемент за id_{i+n}. В случае если такой элемент не найден, вставляем элемент в конце списка. Рассмотрим на примере:

1. $A = a_{0x}x_{3a}b_{1x}c_{2x}, A(X) = (x)[0x, 3a]$;
2. $B = a_{0x}p_{3b}b_{1x}c_{2x}, B(X) = (p)[0x, 3b]$.

- $m(A, B) = a_{0x}p_{3b}b_{1x}c_{2x} \to (x)[0x, 3a] = a_{0x}p_{3b}x_{3a}b_{1x}c_{2x}$ ($3a < 3b$, следовательно, вставляем x после $3b$)
- $m(B, A) = a_{0x}x_{3a}b_{1x}c_{2x} \to (p)[0x, 3b] = a_{0x}p_{3b}x_{3a}b_{1x}c_{2x}$

Список литературы

1. A comprehensive study of Convergent and Commutative Replicated Data Types / M. Shapiro N. Preguiça C. Baquero M. Zawirski – 2011. – Vol. 46, C. 4–40.
2. Key-CRDT Stores / Nuno Manuel Ribeiro Preguiça. – Junho., 2012. – C. 23–40.
3. Convergent and Commutative Replicated Data Types / M. Shapiro, C. Baquer – 2011. – C. 72–84.
4. Etherpad and EasySync Technical Manual / P. Martischka. – Munich, 2011. – C. 2–9.
5. Operational Transformation in Real-Time Group Editors: Issues, Algorithms, and Achievements / C. Sun E. Clarence – School of Computing and Information Technology – Brisbane, Australia, 2018. – C. 1–9.