REVIEW ARTICLE

Operational transformation in cooperative software systems

Clarence Leung[1]*

[1] School of Computer Science, McGill University, Montreal, QC

*Email Correspondence:
clarence.leung@mail.mcgill.ca

Abstract

Modern cooperative software systems involve multiple concurrent users undertaking a common task in a real-time distributed environment, such as editing a shared text document. Maintaining data consistency, transaction causality, and replication convergence in such an environment, while providing fast client responsiveness, is a substantial challenge for classical distributed computing techniques. Operational transformation (OT) is a class of concurrency algorithms and data models that supports these functionalities, which has drawn signicant research attention in the past decade. In this review, we discuss the basic components of operational transformation models, the algorithms involved, and their actual implementations in real-world networked systems. We compare several existing OT control algorithms, the transformation functions and properties supported by each of the algorithms, and the trade-offs that are made with respect to each one. The data and operational models used in OT are well suited for high-latency environments such as the Internet, making them more frequently used in modern web services. Although many different OT control algorithms exist, choosing the most effective one often depends on the particular operations that an application must support.

Introduction

Research in computer-supported cooperative work (CSCW) in the past decade has led to the development of many new collaborative web services, such as blogs, wikis, and social networks. These services can span from only a few simultaneous clients over a local network, to millions of daily users worldwide. Handling the availability of these services becomes more difficult at a larger scale, especially in modern web applications, where notifications of user actions must be delivered in near real-time (1, 4, 7).

However, a common problem that arises when engineering real-time applications, such as collaborative editing services like Google Docs, is to find the appropriate way to handle conflict resolution due to multiple concurrent users. Traditional (pessimistic) approaches to concurrency prevent conflicts from occurring by locking a resource while a client is accessing it, but doing so leads to slower application responsiveness as multiple users are unable to modify a document simultaneously. Other approaches to concurrency, such as optimistic concurrency techniques, assume that data contention is rare, and eschew locks in favour of repairing conflicts when they occur. However, in real-time applications, conflicts are common, and thus simple optimistic concurrency techniques cannot be used without hurting performance while repairing broken states.

A new class of optimistic (non-locking) algorithms and data structures, known as operational transformation (OT), has been highly researched in the past decade in solving these conflict resolution problems. OT algorithms function by considering all user-submitted actions as units known as "operations", and handle any operations that conflict with each other by changing their properties relative to the changes that have already been applied to a particular document.

In this paper, we formally discuss the characteristics of real-time web applications, as well as the models required for us to properly verify that our documents are consistent. Using these models, we describe the operational transformation approach in detail, and survey several different approaches to OT that have been developed.

Traditional web approaches

Traditional web services handle client concurrency through the basic request/response model provided by HTTP, an asymmetric protocol. Request/response paradigms work by not having any response until a client specifically initiates a request to a web server, so no information is passed directly from the server to the client even if new updates have occurred on the server (7).

This protocol would be successful in applications that did not require real-time delivery of user actions, which most blogs, wikis, and representational state transfer (RESTful) web applications would be considered. However, for modern web applications that require real-time collaboration, there are several challenges that need to be overcome. Data must often be broadcasted from the server to the client, to keep the results of user actions up to date. This "server push" can be simulated through continuous polling of the HTTP resource, or keeping a live client-server connection, but this often results in a fragile connection, as it uses the HTTP protocol in a way that it was not intended to be used. Beyond the problems associated with response time, traditional pessimistic concurrency schemes, where a document is locked while one user is updating it, are often poor for developing real-time applications due to a lower responsiveness (2, 3).

Collaborative editing systems

One of the more well-known applications of real-time web services is those of collaborative editing systems such as Google Docs or Etherpad (14). These services provide traditional word processing and spreadsheet applications, but through a shared interface that allows simultaneous editing of a document.
There are three main characteristics that define a collaborative editing system (6):

1. **Real-time:** The local client interface should have a fast response time. Optimally, it should have a response time similar to a single-user editor.

2. **Distributed:** Each user may be geographically separated, from different machines to different communication networks, with varying latency delays.

3. **Unconstrained:** Any user can freely edit any part of the document at any time, without any constraints.

Instead of a pessimistic concurrency scheme, collaborative editing systems often use an optimistic concurrency scheme, where a non-locking transaction scheme is used. Replication is often used in order to allow the local client interface to respond quickly, by having local actions immediately update the local copy of the document. These factors provide the necessary responsiveness for real-time applications, but now lead to a new difficulty: maintaining consistency and properly resolving conflicts throughout the document, while multiple clients are concurrently editing it (8, 9).

Optimistic concurrency techniques

Optimistic concurrency techniques have been well-studied within the context of databases and distributed simulations, which make the assumption that data contention between two different transactions are rare, and conflicts rarely happen. These techniques are often used through web user interfaces, as the stateless nature of HTTP makes it difficult for locks to be used, since a user can simply leave a web page at any time without notifying the application that they have cancelled their transaction.

The simplest form of optimistic concurrency control commits a transaction only when all necessary changes have been tentatively completed, and no conflicts from other transaction have occurred. If a conflict has occurred, a "rollback" is used to revert a data object back to a previous, non-conflicted state, and the transaction which attempted to commit is annulled. This guarantees that our data is always valid and consistent, though this simple form of optimistic concurrency will not be successful in areas where many conflicting operations may occur. The lessened overhead due to locks being unnecessary, however, can lead to greater throughput if the number of conflicts is few.

However, for collaborative editing systems, conflicts occur often, making the simple optimistic concurrency technique infeasible. In later sections, we will discuss the modifications that operational transformation makes to the simple optimistic concurrency technique that make rollbacks unnecessary for resolving conflicts.

Consistency models

Consistency models define the rules that determine what data each client can see in memory, when a particular set of operations have been performed by different clients. An example of when an inconsistency can arise is when one client writes to its own local copy of data in a distributed data store, but has not yet propagated those changes over to the other clients. If those other clients wish to read or write to that data, then they may be using an older version of the data that has not taken into account the other client's modifications.

For collaborative editing systems, several consistency models have been defined to formally verify the properties that clients must support to successfully resolve conflicts in the local copies of their documents. We discuss one of the more commonly referenced models by Sun *et al.* (12), which suggests that consistency in a cooperative editing system be defined by three properties based on each of the operations (discrete transactions submitted by each client):

1. **Convergence:** When the same set of operations has been executed at each site, then the copies of the document are also identical.

2. **Causality preservation:** Given two operations O_A and O_B if $O_A \rightarrow O_B$ (O_A causally occurred before O_B), then O_A is executed before O_B at each site.

3. **Intention preservation:** For every operation *O*, the intention of *O* at the initial site where *O* is initially submitted will be identical to executing *O* at all other sites. The intention of an operation *O* is defined as the resulting document which is achieved by applying *O* on the document state from which *O* was generated.

The convergence property guarantees that the document will be correct at the end of any particular editing time period, while the causality preservation property guarantees that the document will be correct at any point during editing. The intention preservation property is similar to the convergence property, but additionally deals with the situation where operations are submitted from two different initial document states. This property is often the most difficult to achieve, and often requires a submitted operation by one client to be modified once received by another.

Operational transformation

Operational transformation (OT) is a class of optimistic concurrency algorithms and data structures that are well-suited to satisfying the three properties of the consistency model of collaborative editing systems discussed above. The OT mechanism can be divided into the models that represent the data and its changes, and the algorithms that are used to ensure the correctness of the system (1, 3, 4, 5).

Basic example

The basic operational transformation technique (inclusion transformation) can best be illustrated by a simple example. Say we are given a text document, containing the sentence:

"operational transformation is very fun"

Now, two users, A and B, each simultaneously submit an operation on this document. For the purpose of demonstration, an operation has the signature *Operation(position, string)*, where "position" represents the index of a word in the sentence, and "string" is the target string we want to operate on:

1. User A submits the operation O_A, *Insert(0, 'using')*, inserting the word "using" at the beginning of the sentence.

2. User B submits the operation O_B, *Delete(4, 'very')*, deleting the word "very" from the 4th word of the sentence.

If we perform these operations in the order above without transforming them, we get the following result:

"using operational transformation very fun"

As the second operation deletes the 4th word, which accidentally deletes "is" instead of "very". Instead, we need to transform the positional parameter of O_B by one in order to obtain the correct result, which our OT transformation algorithm can calculate and perform, giving us *Delete(5, 'very')* instead. This gives us the correct result:

"using operational transformation is fun"

Data models

Data models in OT are the representations of the data in a particular collaborative session. A data model can represent a document, a spreadsheet, a drawing, or any other object that different clients can collaboratively operate on. The most basic OT data model is that of a linear space in memory, such as that of a single string (1).

Recent developments in OT have allowed the development of more complex OT data models, such as those involving rich-text editing with markup or an advanced application state represented in JSON (JavaScript Object Notation). However, as the data models used in OT become more complex, so do the models that are necessary to represent a particular operation on the data.

Operation models

An operation is the fundamental block of the OT mechanism, and in the simplest OT data model, an operation is simply an insertion or deletion of a single character in memory. As well, more advanced operations can be created from combining insertions and deletions, such as a substitution operation (a combination of an insertion and deletion operation) (10, 11).

For advanced data models that may use more than a linear space in memory, we often find more complex operations such as moving data between memory addresses. For example, OT word processors can add operations to start and close annotation boundaries to represent markup tags, and OT JSON structures can have edit operations on particular lookup paths of a JSON object.

Past research attempted to generalize OT data and operation models, and to only allow insertion, deletion, and substitution operations on the data. However, for most modern OT implementations, application-specific data and operation models are used. These also require operation transformation functions to maintain the intention preservation property, one for each pair of operations, as each type of operation must be able to be modified by the OT control algorithm relative to each other operation type. Thus, for *N* application-specific operations, *N x N* transformation functions are required for OT (13).

Operation composition

For each client in the collaborative system, each client must perform three different activities (12):

1. **Operation generation:** an operation is specified by the user, and broadcast to all of the other clients.
2. **Operation reception:** an operation request is received from another site.
3. **Operation execution:** an operation request is executed at the site.

From these activities, we have two major sets of algorithms that handle conflict resolution for the system: the low-level operation transformation algorithms, and the high-level control (or integration) algorithms. The control algorithms handle the operation models at generation and reception, by generating a set of corresponding transformation algorithms, which then transform the target operation model before executing it at the received site (6).

We will discuss the specific actions taken during each of the activity phases when we compare the different control algorithms possible in OT.

Operation transformation

Operation transformation relies on satisfying several properties that ensure that the system is correct at each point. These are lower-level properties than the consistency model described previously, but not all properties are satisfied by each control algorithm, and where these properties are ensured may differ between each one. These transformation properties are often divided up into two different types: convergence properties and inverse properties (1, 2).

If an OT system is to support particular functionality, then it must be able to support certain transformation properties. For group editing and consistency maintenance, the system must support a transformation function known as Inclusion Transformation (IT). For group undo, where the effect of a previously executed operation is un-done at all sites, and all operations executed after it are all re-transformed, the system must support another transformation function known as Exclusion Transformation (ET).

Transformation functions

As mentioned before, the two base transformation functions that generate all other transformation functions are (11):

1. Inclusion Transformation [$\mathrm{IT}(O_A, O_B)$]: transforms operation O_A against another operation O_B in such a way that the impact of O_B is effectively included.
2. Exclusion Transformation [$\mathrm{ET}(O_A, O_B)$]: transforms operation O_A against another operation O_B in such a way that the impact of O_B is effectively excluded.

As previously mentioned, these transformation functions are generated for each pair of operations in the application space.

The generated transformation functions are basically generated such that the position in which the operation is applied is transformed by an offset relative to the other operation. For example, one of the most basic inclusion transformations between two insert operations can be represented as follows.

For an insert function with signature *insert(p, c, s)* where:

- *p* = position of the character to be inserted
- *c* = character to insert
- *s* = site priority of the client

Then, the transformation function is the algorithm:

Algorithm 1 - Inclusion transform with two inserts
```
if ($p_1$ < $p_2$) then
        return insert($p_1$, $c_1$, $s_1$)
else if ($p_1$ == $p_2$) and ($s_1$ < $s_2$) then
        return insert($p_1$, $c_1$, $s_1$)
else
        return insert($p_1$ + 1, $c_1$, $s_1$)
end if
```

Thus, depending on the priority and insertion location of the operation, we can transform the resulting operation as needed.

Transformation properties

Convergence properties must generally be satisfied by all control algorithms, as part of the convergence guarantee in the consistency model. There are generally two convergence properties, known as CP1 and CP2 (8, 9):

1. **CP1:** Given a state S, and two operations O_a and O_b:

   If $O'_a = \mathrm{IT}(O_a, O_b)$ and $O'_b = \mathrm{IT}(O_b, O_a)$
   Then: $S \circ O_a \circ O'_b = S \circ O_b \circ O_a$

2. **CP2:** Given a state S, and three operations O, O_a, and O_b:

   If $O'_a = \mathrm{IT}(O_a, O_b)$ and $O'_b = \mathrm{IT}(O_b, O_a)$
   Then: $\mathrm{IT}(\mathrm{IT}(O, O_a), O'_b) = \mathrm{IT}(\mathrm{IT}(O, O_b), O'_a)$

Similarly, there are three inverse properties that need to be satisfied to support the "group undo" operation. This is not a part of the consistency model above, but is a commonly implemented operation in practice (8, 9).

1. **IP1:** Given a state S, and sequence of operations $O \circ \overline{O}$:

   $$S \circ O \circ \overline{O} = S$$

2. **IP2:** Given any operation O, and a pair of operations O_x and $\overline{O_x}$:

   $$\mathrm{IT}(\mathrm{IT}(O, O_x), \overline{O_x}) = \mathrm{IT}(O, \mathrm{I}) = 0$$

3. **IP3:** Given a state S, and two operations O_a and O_b:

   If: $O'_a = \mathrm{IT}(O_, O_b)$, and
   $O'_b = \mathrm{IT}(O_b, O_a)$, and
   $\overline{O_a}' = \mathrm{IT}(\overline{O_a}, O'_b)$
   Then: $\overline{O_a}' = \overline{O'_a}$
   Or: $\mathrm{IT}(\overline{O_a}, O'_b) = \overline{\mathrm{IT}(O_a, O_b)}$

Control (integration) algorithms

The OT control algorithm is the main high-level algorithm governing the collaboration functions that are available to the system. This algorithm controls the time/space complexity of the system, handles ordering of the operations by how timestamps are applied, and processes the incoming operations into the modified transformed operations (10, 12).

Most OT systems, by design, are peer-to-peer distributed systems, but modifying the control algorithm can also determine whether the system can also be centralized, which is important when adapting an OT system over the HTTP protocol.

Comparison of control algorithms

We now compare several implementations of OT control algorithms, from ones that are historically relevant in literature to modern day OT control algorithms that are used over HTTP.

dOPT (GROVE)

dOPT, used in the GROVE groupware outline editor program, is one of the earliest concurrency control algorithms for operational transformation (1). It uses a transformation matrix to handle conflict resolution, where for *m* operations, there is an *m x m* matrix of the inclusive transformation resultant functions, for each pair of operations.

Timestamps for each client are handled by a vector timestamp, where a state vector s_i for a client C_i will have at position *j*, the number of operations known to have been executed by client C_j.

A request queue, Q_i, is used to queue up the requests that have been generated or received, and are waiting to be executed. Requests are handled in the form<*j, s, o, p*> where:

- *j* = the site that requested the operation
- *s* = the vector timestamp for the requested site
- *o* = the operation to be performed
- *p* = the priority of the operation

Then, for the three activities (generation, reception, execution), dOPT performs the following:

Algorithm 2 - Generate Operations

```
receive operation $O$ from the user
calculate the priority $p$ of $O$
append request $< i, s, o, p >$ to $Q_i$
multicast $< i, s, o, p >$ to the other clients
```

Algorithm 3 - Receive Operations

```
receive $< j, s, o, p >$ from the network
append request $< j, s, o, p >$ to $Q_i$
```

Algorithm 4 - Execute Operations

```
for each request in $Q_i$ where $s_j \leq s_i$ do
  remove request $< j, s_j, o_j, p_j >$ from $Q_i$
  if $(s_j < s_i)$ then
      $< k, s_k, o_k, p_k >$ = most recent log entry
      where $s_k \leq s_j$ (or Ø otherwise)
      while $< k, s_k, o_k, p_k > \neq$ Ø; and $o_j \neq$ Ø; do
        if part $k$ of $s_j$ is $\leq$ part $k$ of $s_k$ then
          $o_j$ = transform($o_j, o_k, p_j, p_k$)
        end if
        $< k, s_k, o_k, p_k >$ = next log entry
        (or Ø otherwise)
      end while
    end if
  end for
  perform operation $o$ on $i$'s data model
  add request to history log
increment $j^{th}$ component of $s_i$ by 1
```

Although dOPT is simple and satisfies many of the correctness properties, a scenario was found where dOPT could not always ensure convergence, when remote concurrent requests with similar operations were transmitted from two different sites.

Further research later helped to solve the problem, by using different data structures for the timestamp and conflict resolution (as in the Jupiter algorithm), or by transforming the log entries themselves whenever they are used to transform an update.

Jupiter

Jupiter, a multi-user remote collaboration virtual world developed at Xerox PARC by Nichols *et al.*, addressed some of the issues found in dOPT (7). One of the major changes in Jupiter was to have a centralized coordination server that uses a change propagation algorithm to keep the clients updated and in check.

The optimistic concurrency techniques are used in the individual client-server links, where the synchronization algorithm in the individual client-server links is very similar to the dOPT algorithm.

The major changes in Jupiter, in comparison with dOPT, is that instead of a transformation matrix, Jupiter uses a function called *xform*, which takes a pair of client and server operation requests, and transforms them as a new pair of operations that lead the client and server operations to the same final state.

The *xform* technique is successful while the client and server are in the same starting state, but if the client and server diverge too much, then the operation must look back into its past history to properly calculate the converging operations. This is often done by keeping track of operation revision history, but not directly transforming saved messages, like dOPT does, which causes the incorrect scenario. Instead, Jupiter calculates past converging operations, even if it does not apply them, in order to generate the correct recent transformed operation.

Google Wave OT

Google Wave OT, the algorithm behind Google Docs and the former Google Wave product, was a modification on top of the Jupiter algorithm (14). The main changes to the Jupiter control algorithm are mainly in the client/server communication protocol, as well as optimizing the transformation functions for batch updates.

Instead of sending client operations to the server whenever new operations are requested, the Google Wave OT client must wait for a server response before sending any more operations. In the meantime, the operations are composed together into a single buffer, and then sent together once the server has processed the last batch of operations and converged.

This technique serves two major purposes. First, as the server acknowledges the client before any new operations proceed, the client is capable of predicting the operation path that the server will take, and thus always send operations to the server that are always on the path. This simplifies the server implementation significantly, as the server only needs to keep track of its own history. Instead of taking a quadratic $O(h^2)$ space for history on every possible path, it only takes up a linear $O(h^2)$ space on the server for history, where h is the number of previous operations (and possible causal operations) required to calculate the correct OT path.

Finally, the transformation algorithm handles operations in streams, rather than as single discrete operations. These operation sequence streams are guaranteed to be in order, as well as linear, so the streaming transformations can be performed in linear time.

ABT (TIPS)

A more modern protocol for OT is based on the admissibility-based transformation (ABT) framework, which has been formally verified for correctness (8,10). TIPS, an implementation of ABT, builds on top of existing HTTP protocols, and also uses a centralized server.

Clients in TIPS are able to join or leave a session at any time, and the clients also independently decide to sync with the server. As with Google Wave OT, operations are buffered and synchronized with the server only at some particular server-determined interval. This is useful, as it can be easily adapted to JavaScript-based long polling methods, which are the traditional method of maintaining an open connection over the web.

Once the server has asked all of the clients to synchronize, each client that has responded will have their operations applied to an *n*-way merge algorithm, which will create a sequential set of operations as output. The clients receive this sequential set of operations in another interval, which is adapted by them through another algorithm, ITSQ, which performs the inclusion transformation step that updates the document correctly.

The benefit of TIPS over existing protocols is that it is more capable of allowing clients to dynamically enter and exit a particular OT session, as well as being more robust when dealing with client or network failures.

Summary

Operational transform is still a rapidly developing technology, with many particular algorithms and system implementations that are well-suited for particular tasks. These algorithms can differ in the consistency guarantees they support, as well as functionality, and the most useful algorithm often depends on the specific application that is being built. Both the high-level design of the OT control algorithm, as well as the low-level design of the OT data and operation models is significant in determining which systems are the most practical.

Real-time collaborative applications are slowly becoming some of the more widely used applications in business environments, and with further development with Internet-accessible mobile devices, these applications will continue to flourish. Operational transformation will continue to be a significant research topic in the coming years as networked cooperation becomes more widely used.

Acknowledgements

I would like to thank Eric Ferraiuolo of Yahoo! for assisting me in learning about the field, and for helping to provide resources and technical advice.

References

[1] C. A. Ellis and S. J. Gibbs. *Concurrency control in groupware systems.* In ACM SIGMOD89 Proceedings, **18**(2):399407, 1989.
[2] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. *Achieving convergence, causality-preservation, and intention preservation in real-time cooperative editing systems.* ACM Transactions on Computer-Human Interaction, **5**(1):63108.
[3] R. Bentley and P. Dourish. *Medium versus mechanism: Supporting collaboration through customization.* In ECSCW '95 Proceedings, 1995.
[4] A. H. Davis, C. Sun, and J. Lu. *Generalizing operational transformation to the standard general markup language.* In ACM CSCW'02, pp. 58 - 67, Nov. 2002.
[5] G. V. Cormack. *A calculus for concurrent update.* Technical Report CS-95-06, Dept. of Computer Science, University of Waterloo, Canada, 1995.
[6] D. A. Nichols, P. Curtis, M. Dixon, and J. Lamping. *High latency, low-bandwidth windowing in the Jupiter collaboration system.* In ACM UIST 95 Proceedings, Nov. 1995.
[7] B. Shao, D. Li, T. Lu, and N. Gu. *An operational transformation based synchronization protocol for Web 2.0 applications.* In CSCW 2011, pp. 563 572, Mar. 2011.
[8] D. Li and R. Li. *An approach to ensuring consistency in peer-to-peer real-time group editors.* Computer Supported Cooperative Work: The Journal of Collaborative Computing, **17**(5-6):553-611, Dec. 2008.
[9] D. Li and R. Li. *An admissibility-based operational transformation framework for collaborative editing systems.* Computer Supported Co-operative Work: The Journal of Collaborative Computing, **19**:1-43, Aug. 2009.
[10] R. Li, D. Li, and C. Sun. *A Time Interval Based Consistency Control Algorithm for Interactive Groupware Applications.* In Proc. IEEE Intl Conf. Parallel and Distributed Systems, pp. 429-436, Jul. 2004.
[11] C. Sun and C. Ellis. *Operational transformation in realtime group editors: issues, algorithms, and achievements.* In Proceedings of the ACM Conference on Computer-Supported Cooperative Work, pp. 59-68, Dec. 1998.
[12] N. Vidot, M. Cart, J. Ferrie, and M. Suleiman. *Copies Convergence in a Distributed Realtime Collaborative Environment.* In Proceedings of the ACM Conference on Computer-Supported Cooperative Work, pp. 171-180, Dec. 2000.
[13] D.Wang, A. Mah, and S. Lassen. *Google Wave operational transform.* Google Wave Federation Protocol White Papers, July 2010.