Introducing CEDA

David Barrett-Lennard
david.barrett-lennard@thinkbottomup.com.au

November 2009

Abstract:

CEDA is a high performance database technology that uses Operational Transformation (OT) to support replication and synchronisation for collaborative data entry performed by multiple users. CEDA is an ideal platform for Computer Supported Collaborative Work (CSCW), allowing multiple users to edit that same data concurrently. Users can collaborate in a interactive, realtime manner or work offline and control when changes are propagated to/from other users.

CEDA provides an application development platform for building applications that support collaborative work. Application developers can easily build multiuser applications based on CEDA that are as responsive as single user applications while attaining benefits of realtime interactive collaboration and sophisticated version control. The platform allows for branching and merging of the entire database, supporting manual check-ins, check-outs, updates, tagging and so forth in a similar manner to source code repository systems like ClearCase and Subversion.

The OT algorithms employed by CEDA are extraordinarily efficient, allowing users to work off line for long periods of time and then quickly synchronise with other users.

This paper provides an introduction to CEDA and some of the platforms features.

Think Bottom Up Pty Ltd
ABN 85 130 346 119

16/9 Vale St, Malaga WA 6090
E info@thinkbottomup.com.au
T +61 8 6460 1643 F +61 8 6210 1175
www.thinkbottomup.com.au

ThinkBottomUp

Overview

CEDA is an ideal platform for Computer Supported Collaborative Work (CSCW). It is essentially a high performance database technology that uses Operational Transformation (OT) to support replication and synchronisation for collaborative data entry by multiple users.

Each site persists its own copy of the data in a local database and changes (called operations) are always applied immediately on the site where they are generated - i.e. independently of network latency. The principle is that multi-user applications can and should be as responsive as single user applications. It also allows users to continue working when there are network outages.

A group of sites can be configured to support real-time editing of the replicated data. In that case the operations are exchanged between sites asynchronously in the background in order to synchronise the databases. This can easily happen at the rate at which users type on the keyboard or manipulate objects with the mouse. Applications are highly interactive, even on networks with high latency and low bandwidth.

CEDA is not restricted to real time collaboration. It also allows users to work in isolation and control precisely when changes are merged. The platform allows for branching and merging of the entire database, supporting manual check-ins, check-outs, updates, tagging and so forth in a similar manner to source code repository systems like ClearCase and Subversion.

Jigsaw demo application

A simple jigsaw demonstration application has recently been developed that allows hundreds of users to work on a shared jigsaw. A given user may see dozens of jigsaw pieces all moving in real time as operations are received from other sites.

The use of OT is apparent in two ways:

- A user can always manipulate the position of a jigsaw piece without latency.
- A user is able to continue working if the network becomes partitioned. When network connectivity is re-established, many jigsaw pieces may suddenly move as databases resynchronise.

This application has the remarkable property of allowing users to work offline for hours on a 5000 piece jigsaw, and the changes can be merged in less than a second.

Operational Transform

OT avoids the need for distributed locking and distributed transactions, by applying operations in different orders at different sites. It was pioneered 20 years ago and is described in Wikipedia,

See http://wikipedia.org/wiki/Operational_transformation.

The key idea is that operations sent to a remote site are transformed as required so they achieve the equivalent effect where they are applied.

In Figure 1, operations O1, O2 are executed at sites 1 and 2 respectively. It is assumed the sites are initially in the same state S, and applying the operations leads to divergent states S1 and S2. The operations are sent over the network and are transformed to O1’ and O2’ so as to preserve the original intentions of the operations and also to ensure the two sites converge to the same final state, even though they executed the operations in different orders.

We can depict this with the state transition diagram shown in Figure 2. O2’ = IT(O2,O1) denotes the Inclusion Transformation of O2 with respect to O1. It is the transformed version of O2 that has now included O1 in its execution context.

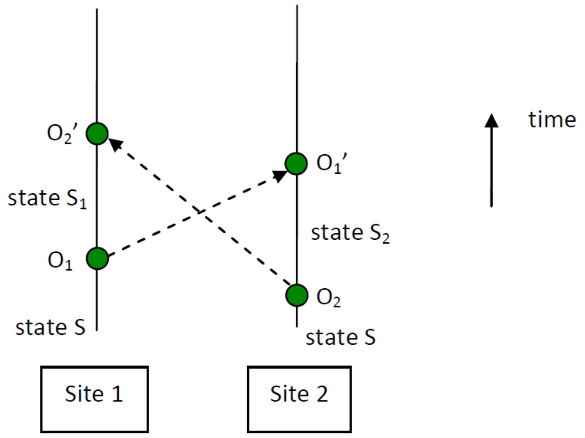


Figure 1: Operational Transformation

For example, the insertion position of an operation recording an insertion into a text field may be shifted as a result of concurrent operations to the left of the insertion position. Here, concurrent means operations that were not already in the original context of the insertion at the time it was generated.

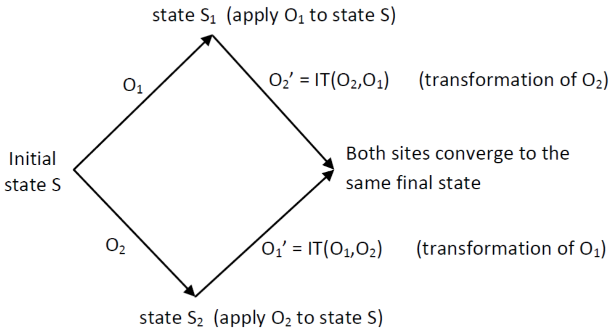
Figure 3 depicts a more specific example. O1 and O2 are respectively insert and delete operations on a shared text document initially containing the text "abc". In this example the delete position of O2 needs to be shifted by two characters to the right as O2 is transformed to include the effect of O1.

This one example doesn't convey the complexity of the problem and the subtle issues that have led to numerous erroneous solutions in the literature. When we generalise from two sites to three, we end up with a cube where one corner is associated with the initial state from which 3 concurrent operations O1, O2, O3 make changes (metaphorically diverging in 3 different directions). The aim is to transform operations on the faces of the cube in such a way that convergence is achieved on the opposite corner of the cube. There are 6 distinct paths that can be take. Two of the six paths are depicted in Figure 4.

Convergence requires the following transformation property (called TP2 in the literature) to be satisfied:

$$IT(IT(O3, O1), IT(O2, O1)) = IT(IT(O3, O2), IT(O1, O2))$$

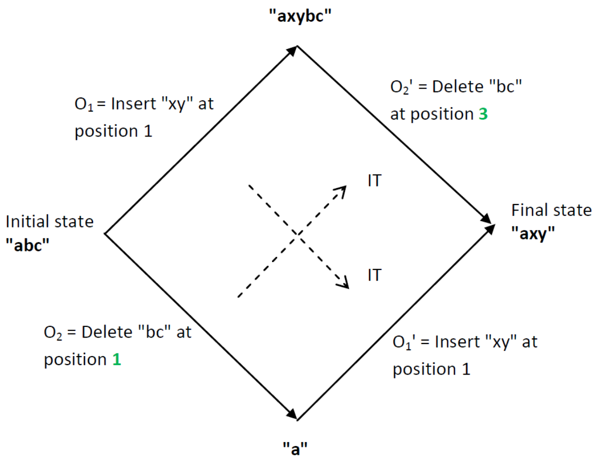
It turns out that convergence on each face is insufficient to ensure convergence on the cube. In the literature such counterexamples are referred to as TP2 puzzles (because many published solutions fail to meet TP2 and therefore may fail to converge for 3 sites or more).



state S_1 (apply O_1 to state S)
O_1
$O_2' = IT(O_2,O_1)$ (transformation of O_2)
Initial state S
Both sites converge to the same final state
O_2
$O_1' = IT(O_1,O_2)$ (transformation of O_1)
state S_2 (apply O_2 to state S)

Figure 2: State Transition Diagram

Somewhat surprisingly a correct implementation of OT allows concurrent edits to always be merged with zero "syntactic" conflicts. For example, in the case of simple text, the merge result is defined to be the union of all the insertions minus the union of all the deletions. This is only ambiguous in the order of insertions at exactly the same position. The system resolves this ambiguity using some arbitrary total ordering on the sites, and users always find the merging to be faithful because every insertion and deletion has been applied.



"axybc"
O_1 = Insert "xy" at position 1
O_2' = Delete "bc" at position 3
IT
Initial state "abc"
Final state "axy"
IT
O_2 = Delete "bc" at position 1
O_1' = Insert "xy" at position 1
"a"

Figure 3: Operational Transformation Example

Multicast, Vector Clocks and Topology

CEDA allows for an arbitrary topology of connections between computers, and will ensure operations are exchanged as required so that all computers eventually receive all operations exactly once.

An operation can be generated by any computer in the network, and the operation is automatically multicast to all other computers (note this doesn't involve IP multicast). This is very efficient because an operation will be sent across a given link at most once.

It is even possible for any number of computers to "crash" and roll back to an earlier state (by virtue of transaction atomicity of each local database) and reconnect in an entirely different topology. A computer will then receive all missing operations to bring it up to date. This can even include operations that originated on that computer before it crashed!

The system avoids the need for distributed transactions, and all the associated complexity of multiphase commit protocols. Instead it only requires simple transient message queues for all communication between sites.

Any two machines can connect at any time and always successfully exchange operations and synchronise - even if they have never connected in the past.

Operations are always applied in an order consistent with the partial ordering associated with causal relationships between operations. This involves the use of Vector Clocks which are described, for example, in the classic “Time, Clocks and the ordering of events in a distributed system”, Leslie Lamport (1978).

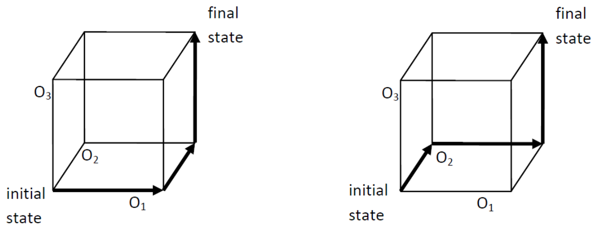


Figure 4: OT Cube Paths

OT Efficiency

According to the literature large networks of replicated databases are not practical due to scalability problems. This is certainly true for conventional approaches – for both pessimistic and optimistic locking systems. Jim Gray (a highly regarded researcher in database systems) found that the number of deadlocks or reconciliations grows with cubic order on the size of the system. See The Dangers of Replication and a Solution, 1996 Microsoft Research. In practise this could mean a machine takes minutes or even hours merging off-line work.

Figure 5 is relevant for two sites that are initially in the common state S and need to inclusion transform lists of operations L1 and L2.

The inclusion transform of complex operations can always be broken down into the inclusion transform of simple operations. Evidently the naive approach is quadratic in the size of the lists.

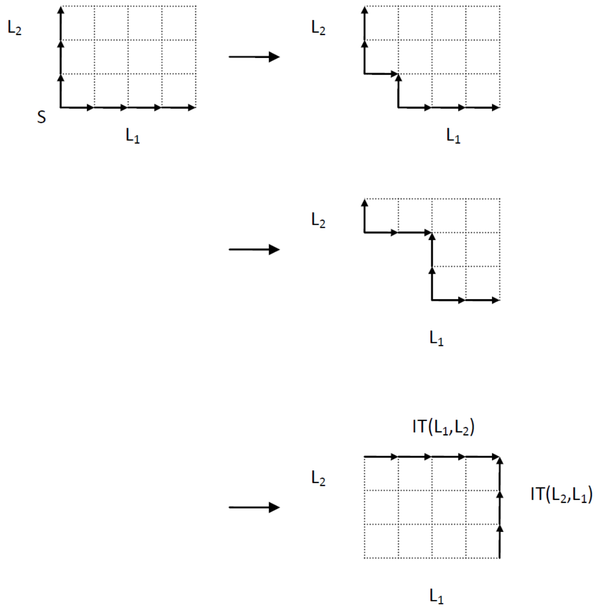


Figure 5: Transforming Lists of Operations

CEDA uses various proprietary algorithms and techniques (that in particular avoid this quadratic complexity, and also avoid some assumptions made in Gray's paper) making it extraordinarily efficient.

Log Structured Store

The database system uses a proprietary Log Structured Store suited to the persistence of complex data structures. The store supports recovery, backup, hot standby and is self cleaning to avoid fragmentation. It provides excellent control over clustering to optimise read performance, and the ingestion rate closely matches the sustained write rate of a hard-disk (e.g. 100 MB/sec), and yet provides proper journaling for atomicity in the face of failures. A client (Runge Limited) using this technology said they found it substantially faster than BTrieve for their application.

Dependency Graph System

The framework provides a Dependency Graph System used to efficiently recalculate dependents satisfying one-way constraints. This is well suited for GUI development because the dependencies between visual elements and the underlying persistent data is discovered and managed automatically. This eliminates an enormous burden from the application programmer.

This framework automatically detects cycles in the graph and early quiescence (i.e. where dependent nodes haven't changed in value, downstream dependent nodes can avoid unnecessary recalculation).

Database Schema

A key concept is the definition of a database schema in a purposely designed language. For example, the following code is taken from the jigsaw demonstration application:

```
$tuple TPoint2d
{
        int32 X;
        int32 Y;
};

$tuple TColour
{
        uint8 R;
        uint8 G;
        uint8 B;
};

$tuple TJigsawPiece
{
        TPoint2d Position;
        bool JoinedOnRight;
        bool JoinedOnBottom;
};

$tuple TJigsaw
```

```
	{
		string8 Username;
		TColour BackdropColour;
		TPoint2d DeskTopSize;
		int32 PixelTolerance;
		int32 GeometryFactor;
		int32 RandomNumberSeed;
		int32 NumRows;
		int32 NumCols;
		TJigsawPiece Pieces[];
	};
```

These data types automatically support persistence and replication using OT. Note as well that CEDA provides very advanced support for schema evolution - i.e. allowing for these data types to evolve over time.

The following code shows some examples of how the programmer can mutate the data directly in C++:

```
	js->NumRows = 10;
	js->NumCols = 12;
	js->Pieces[3].Position.X = 100;
	js->Pieces[3].Position.Y = 200;
```

These assignments are automatically persisted and synchronised to other sites.

A reflection system is used that allows the above data types to be mapped directly to C++ data types, and yet be accessed by other languages like Python though the reflection information.