Ceda concepts

David Barrett-Lennard

11 Oct 2016

# Introduction

CEDA is a high performance database technology suitable for the management of all sorts of data, such as

* spatial data for mining software
* CAD drawings
* word processing documents.
* multi-resolution terapixel images

The perfectly smooth panning and zooming of images on the scale of Google Earth on a low performance laptop highlights the exceptional performance characteristics of the database engine.

CEDA can scale from one user to many thousands of users which interactively edit very complex and large amounts of data in applications that feel as responsive as single user applications. The data is replicated and synchronised efficiently, and users are able to view and edit their local copy of the data independently of network latency and network failures. This is achieved using Operational Transformation (OT). The unique and revolutionary algorithms in CEDA typically merge hours of off-line data entry in a fraction of a second.

CEDA is implemented in C++, and provides a front end (called Xcpp) for some language extensions to C++. In particular it allows for defining data types that support the following:

* Reflection (i.e. type information is recorded about the data)
* Persistence in a database
* Replication and synchronisation
* Read and write access from Python

# Log structured store

CEDA uses a Log Structured Store to persist data on secondary storage. It achieves read/write performance unmatched by other database technologies. It has been found to outperform BTrieve by a significant factor and yet Btrieve is supposed to be one of the fastest database systems in the world.

Conventional databases use Write Ahead Logging (WAL) to allow a server process to fail at any time and yet the database is able to recover back to a consistent state when the server is restarted. For that reason it is actually unsafe for products like SQL Server or Oracle to use stock hardware without disabling the hard-disk write caches which can reorder writes (but that is rarely done because write performance would drop by a factor of 10 or more - maybe even 100). By contrast CEDA uses a far more resilient and efficient system for crash recovery that is almost independent of the order of writes and therefore allows for crash recovery to be performed without needing to disable the write cache on the hard-disk.

# Data replication

In this document a site typically means a computer.

CEDA allows data to be replicated meaning each site has its own copy of the data stored in its own local database. Operations (*updates* to the database) are exchanged between sites over the network in order to synchronise the databases.

Site A

Site B

Operations

Database

The sites may be geographically separated, maybe on different sides of the planet.

# The fallacies of distributed computing

One of the prime motivations of CEDA is to support distributed data management, despite the fact that networks tend to be unreliable, have high latency, low bandwidth etc. Programmers new to distributed applications invariably make the following false assumptions

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

# Data redundancy and load balancing

Data replication provides data redundancy and allows for load balancing. A site can lose its copy of the data and receive a copy from its peers.

Data is only truly lost when all sites have lost the data.

# Operations

Data replication depends on the ability to synchronise the replicated data by sending database changes (called deltas or operations). Operations are serialised over a TCP/IP socket connection as a stream of bytes.

The operations can be very fine-grained, meaning for example that operations record changes down to the level of insertions or deletions of individual characters in a text field. Therefore when users interactively edit some shared data they get to see each other’s edits in real time, such as typing in a text field or manipulating objects with the mouse.

Network bandwidth utilisation is excellent because synchronisation involves deltas which only record the changes, irrespective of the total amount of replicated data.

Ceda is efficient enough to allow thousands of users to interactively collaborate, exchanging hundreds of thousands of operations per second.

# Applications only access the local database

Applications that view and edit the data only access the local database. The local access is high bandwidth, low latency and always available. Even though applications are multi-user they feel as responsive as single user applications.

The applications don’t directly communicate. Instead the CEDA distributed DBMS does all the work. Applications just access the local database and don’t have to worry about the network, latency, replication and synchronisation.

Site A

Site B

**App**

**App**

Low latency

High bandwidth

Always available

Responsive to user

Apps don’t directly communicate

# Basic idea of OT

Operational Transformation (OT) allows operations to be applied in different orders at different sites. In the following picture site A generates an operation O1 and concurrently site B generates an operation O2. These operations are applied immediately on the database where they are generated.

Each operation is sent to the peer where it is transformed in such a way that it can be applied in a new context and yet preserves the original intention of the operation. So the two sites execute the operations in different orders and yet they reach the same final state.

O1

O2

Site A

Site B

O2’

O1’

**App**

time

**App**

So to summarise:

* Data is replicated
* Operations are generated and applied immediately on a local database
* Operations are sent to peers
* Received operations are **transformed** and applied in the background
* Operations can be applied in different orders at different sites
* All sites eventually converge to the same state, once they have received all operations

# Multi-master replicated database

CEDA allows for many sites (even hundreds of thousands) in arbitrary network topologies.

The following picture shows an operation generated at one site and propagating efficiently through the network, to be delivered to all sites. At the same time other sites can be generating operations as well, so that at any given time there can be many operations propagating in different directions through the network.

Offline

Generate operation

**App**

**App**

Operations on the same data can be generated at different sites regardless of network partitions. In the database theory literature this is called multi-master replication and is known to be highly desirable but very challenging to implement. Indeed there have been hundreds of papers on the subject in the past 40 years. It is like the Holy Grail for data management systems, especially large distributed systems.

It is also called update-anywhere-anytime replication, because a user is always able to update their local copy of the data, even if they are disconnected from other computers. Indeed the network can be very unreliable, dropping out intermittently all the time, and yet users continue working on their local copy, immune to these problems. The editing experience is unaffected by network latency or disconnections. It means multi-user software is just as responsive as single user software.

Multi-master replication is known to be difficult because of some negative results which have been established in the literature, such as the CAP theorem which shows that it is impossible for a distributed database system to guarantee all three of the following:

* consistency
* availability
* partition tolerance.

In other words any system must pick two and give up on the third. CEDA addresses the limitations of the CAP theorem by allowing sites to temporarily diverge as operations are performed in different orders. This is sometimes called eventual consistency.

Once all sites have received all operations they necessarily converge to the same state. CEDA does not compromise on availability and partition tolerance (in contrast to systems which don’t are therefore are fragile). When there is a network failure users are able to continue updating their local copies of the data, they are autonomous. The algorithms are very robust, and allow for redundant data paths, failed connections, changes to the network topology and numerous other failure conditions.

In fact CEDA is well suited to replication in extremely unreliable networks. It even allows connections to be broken every few seconds and yet allows robust synchronisation of replicated data. This has been proven to work with reconnections in arbitrary network topologies that change over time. Computers can even connect that have never directly connected before in the past and exchange operations that were received from other peers. The CEDA replication system was first implemented 8 years ago and has been subjected to ongoing, heavy testing with billions of randomly generated operations on randomly generated network topologies with randomly generated reconnections.

Another negative result in the literature is a paper showing that under quite general assumptions replication is not scalable because the computational load increases 1000 fold when the system is 10 times larger. This can easily mean a system cannot keep up with a very modest transaction rate, much to the surprise of its developers. Such a situation is unrecoverable because the load increases as the divergence in the copies of the database increases.

As a result many systems tend to only use master-slave replication. This means updates can only be applied to one computer (the “master”) and updates only propagate in one direction to all the “slaves”. This is quite limiting compared to update-anywhere-anytime replication. E.g. users cannot work if they cannot connect to the master and the data entry application may seem sluggish because of network latency (i.e. the time for messages to be sent to and from the master over the network).

Nevertheless CEDA has a computational load which is proportional to the size of the system, possible because it avoids the assumptions in the literature than imply replication cannot scale. In fact the algorithms are extraordinarily efficient.

Google have tried to support update-anywhere-anytime with Google Wave, a project that caught the interest of industry experts for its exciting proposal to use Operational Transformation to achieve multi-master replication, but their solution doesn’t satisfy a mathematical property in the literature called TP2, which means it is not able to merge changes in arbitrary orders for arbitrary network topologies.

CEDA was compared to ObjectStore (a popular object oriented DBMS) in 2004 and CEDA was found to achieve 100 times the transaction throughput in a multi-user database system on a LAN. The benefits of CEDA would have been even greater on a WAN. This is essentially because CEDA uses multi-master replication with fully asynchronous operations, whereas Object Store uses distributed transactions, multi-phase commit and pessimistic locking. ObjectStore is using the conventional approach still emphasised in the database literature today, but which exhibits both poor performance and can’t be made robust to network partitions because of the theoretical impossibility of guaranteeing all sites agree to commit a distributed transaction or not when the network is unreliable.

Keys features include:

* Avoids pessimistic locking of data and the complexity and pitfalls with distributed locking, dead-lock detection, and multiphase commit protocols;
* True peer-peer based replication with a symmetric protocol;
* Locally generated operations are always applied immediately without being delayed or blocked – even in the presence of network failure;
* Efficient: The sender only sends operations that have not yet been applied on the receiver. Operations are only sent over a link once; and
* Robust:
	+ The network can partition at any time in any way and users can continue working;
	+ There can be arbitrary changes to the network topology at any time. Sites can reconnect in an entirely new topology and exchange operations as required;
	+ Any site can fail permanently or temporarily. A site can crash and roll back to an earlier state; and
	+ There can be cycles in the topology

## Operations are streamed between sites

CEDA synchronises databases without distributed transactions.

The sender only reads its local database to enqueue operations in a send buffer. The receiver updates its local database with local atomic non-durable transactions. There is no need to synchronise with the sender because machines don't persist information about what operations have been applied on peers - indeed the receiver could crash and roll back to an earlier state, and when it next connects it may receive operations it had previously received before it crashed.

Flushing (durability) on the receiver is not needed for correctness.

Sender enqueues operations

Transport layer

Receiver dequeues operations

Send Buffer

Receive Buffer

Site A

Site B

# Transactions

All access to the local database (whether for read or write access) must be done inside an explicitly declared transaction.

A transaction defines an atomic change to the local database. If the process crashes or the power fails, the CEDA database will ensure an incomplete transaction is rolled back the next time the database is opened. Crash recovery has been designed to never takes longer than about 1 second.

The data is partitioned into coarse mutually exclusive pspaces and all required locks on the pspaces must be granted before a transaction begins. This is done in such a way that dead-lock is not possible.

# Operational Transformation Semantics

## Assignable variables

Assignment operations are inherently “lossy” in the sense that after many assignment operations to the same variable the resulting value is determined only by the last assignment, all earlier assignments have been lost. As an example, consider a slider control which causes assignment operations on a variable to be generated (typically about 50 per second) as the slider is dragged. Only the final value is preserved.

This lossy nature means it is usually only practical to have assignment operations on very simple data types.

This inherent lossy nature also exists when merging concurrent assignment operations to the same variable. The system must pick one site as the “winner” and assignments from other sites are lost.

# Text

The CEDA merge algorithms allow for arbitrary merging of changes to text. The result is of a merge is always defined, and it is always the union of all the insertions minus the union of all the deletions. The relative order of the inserted characters is always maintained when merging operations.

abcde

O2

O1

O2

O4

O3

Merge result =

 (union of inserts)

 minus

 (union of deletes)

Zero conflicts under merge!

O3

# OT on millions of variables

The data is represented in a hierarchical manner using sets, tuples, maps, arrays and objects. The operations are typically very fine grained and target the “leaf variables” which tend to be simple data types. The leaf variables are assumed to be independently updatable.

v1

v2

v4

v3

v5

v6

v7

v8

v9

v10

v11

v12

sets, tuples, maps, arrays, vectors or objects

Operations typically target the "leaf" variables

For example consider the following data structures

|  |
| --- |
| $model+ Point{ float32 x; float32 y;};$struct+ SomeClass isa IPersistable : model  { xvector<Point> s; xmap<string8, Point[5][4]> m; float64 f[6]; }{}; |

An operation is parameterised by a “path” through the hierarchical representation. In other words by a list of values/identifiers which identify the variable which is the target of the update.

Let the following C++ code be executed where p is of type SomeClass\*

 (\*p).m["a"][3][2].x = 3;

This is an assignment to a variable which is identified by the path

 [p, m, "a", 3, 2, x]

A path is an ordered list of identifiers. This path identifies a heap allocated object (by object reference value), members within models (by index), elements of arrays (by index) and mapped values (by key).