Implementation of CRDTs with δ-mutators

Carlos Baquero

(Joint work with Paulo Almeida, Ali Shoker)

Presented at SyncFree M24 Meeting, September 2015

HASLab
HIGH-ASSURANCE
SOFTWARE LABORATORY

Universidade do Minho

INESCTEC
TECHNOLOGY & SCIENCE
ASSOCIATE LABORATORY
COORDINATED BY
INESCPORTO
PORTUGAL

SYNC FREE

Why Deltas?

- State-based CRDTs
  - ⊞ Simple middleware, Gossip
  - ⊟ Complex/Big state, with UIDs and concurrency info
- Replicas evolve by mutations, inflations in a lattice
  - $X' = m(X)$ e.g. GSet $\{a, b, c\} = add_b(\{a, c\})$
  - $X' = X \sqcup m(X)$
  - Updating other replicas ⇒ shipping the big X'
- $\delta-$mutations
  - $\delta = m^\delta(X)$ e.g. $\{b\} = add_b^\delta(\{a, c\})$
  - $X' = X \sqcup \delta$ $\{a, b, c\} = \{a, c\} \sqcup \{b\}$
  - Updating other replicas ⇒ shipping δ, hoping $\delta \ll X'$
- δ's can be merged in transit. Usually applied in causal order

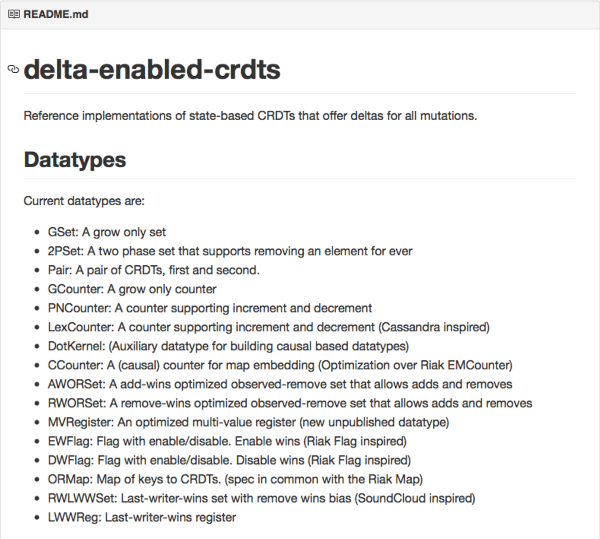
Why C++?

- Existing libraries: Python, Java, Erlang, Akka, (later Elixir)
- Strongly typed approach, no pointers and casts used.
- Good starting point from the Standard Template Library
- Efficiency . . . author already familiar with the language

GitHub

https://github.com/CBaquero/delta-enabled-crdts

Delta Enabled CRDTs



README.md

delta-enabled-crdts

Reference implementations of state-based CRDTs that offer deltas for all mutations.

Datatypes

Current datatypes are:

- GSet: A grow only set
- 2PSet: A two phase set that supports removing an element for ever
- Pair: A pair of CRDTs, first and second.
- GCounter: A grow only counter
- PNCounter: A counter supporting increment and decrement
- LexCounter: A counter supporting increment and decrement (Cassandra inspired)
- DotKernel: (Auxiliary datatype for building causal based datatypes)
- CCounter: A (causal) counter for map embedding (Optimization over Riak EMCounter)
- AWORSet: A add-wins optimized observed-remove set that allows adds and removes
- RWORSet: A remove-wins optimized observed-remove set that allows adds and removes
- MVRegister: An optimized multi-value register (new unpublished datatype)
- EWFlag: Flag with enable/disable. Enable wins (Riak Flag inspired)
- DWFlag: Flag with enable/disable. Disable wins (Riak Flag inspired)
- ORMap: Map of keys to CRDTs. (spec in common with the Riak Map)
- RWLWWSet: Last-writer-wins set with remove wins bias (SoundCloud inspired)
- LWWReg: Last-writer-wins register

Primitive Types

A **join** template function was defined for taking **max** from ordered primitive types: char, int, float, bool, . . .

```
int a=2, b=0;
cout << join(a,b) << endl;  // Output is 2

char x='a', y='b';
x=join(x,y);
cout << x << endl;  // Output is b
```

Pair Composition

STL has a template pair composition. A point-wise join was defined

```
pair<int,char> a(1,'a'),b(0,'x');
cout << join(a,b) << endl;  // Output is pair (1,x)
```

While the point-wise version is default, a lexicographic join was also defined

```
cout << lexjoin(a,b) << endl;  // Output is pair (1,a)
```

Pairs can be nested and include non primitive types

```
pair<int,pair<gset<int>,char>> triplet;
```

GSet, a simple anonymous CRDT

Family: GSet, TwoPSet

Classic use

```
gset<string> a,b;

a.add("red"); b.add("blue");

a=b=join(a,b);

cout << a << endl; // GSet: ( blue red )
```

Obtaining deltas

```
gset<string> d = a.add("green");

b.join(d);

cout << a << endl; // GSet: ( blue green red )
cout << b << endl; // GSet: ( blue green red )
```

PNCounter, a simple identified CRDT

Family: GCounter, PNCounter, LexCounter

Counters can be formed from any number type. Deltas can be anonymous but mutable instances must have a unique id.

```
pncounter<long,char> x('a'), y('b'),d;

x.inc(4); x.dec();
d=y.dec();

x.join(d);

cout << x.read() << endl; // Output is 2
```

Default template types are **int** for counter and **string** for id

```
pncounter<> z("syncfree"); z.inc();
cout << z.read() << endl; // Output is 1
```

Dot Kernel for Causal CRDTs

Family: CCounter, AWORSet, RWORSet, MVRegister, EWFlag, DWFlag

Causal CRDTs, implemented by a kernel type with a universal join.
All supported types are optimized (aka without tombstones)

Kernel Stucture

- DotContext: A version vector plus a sparse dot cloud
- DataStore: Mapping dots to chosen payload values

- Causal CRDTs hold an instance of a kernel.
- Used to add new dot to value pairs, remove pairs, join
- Causal information is grow-only and compacted when possible
- Possible to share a DotContext among instances, in maps

Add-Wins Observed-Remove Set

Family: CCounter, AWORSet, RWORSet, MVRegister, EWFlag, DWFlag

Classic use

```
aworset<float> x("uid-x"), y("uid-y"), d;

x.add(3.14); x.add(2.718); x.rmv(3.14);
d=y.add(3.14); // Concurrent add to above remove

x.join(d);

cout << x.read() << endl; // Output is ( 2.718 3.14 )
```

All kernel types support an *observed reset*

```
x.reset(); x.join(y);

cout << x.read() << endl; // Output is ( )
```

(Optimized) Multi-Value Register

Family: CCounter, AWORSet, RWORSet, MVRegister, EWFlag, DWFlag

Collecting and merging deltas on site x

```
mvreg<string> x(”uid-x”),y(”uid-y”),d;

d=x.write(”hello”); d.join(x.write(”world”));

y.write(”world”); y.write(”hello”);

y.join(d);

cout << y.read() << endl; // Output is ( hello world )
```

(Optimized) Multi-Value Register

Reducing siblings, (joint design with Marek, Nuno, Annette, Marc)

Concurrent values related in an order can be reduced.

Total order example

```
mvreg<int> x(”uid−x”), y(”uid−y”);

x.write(0); y.write(3);
x.join(y); x.resolve();

cout << x.read() << endl; // Output is ( 3 )

x.write(1); // Value can go up and down
```

(Optimized) Multi-Value Register

Reducing siblings, (joint design with Marek, Nuno, Annette, Marc)

Partial order example

```
mvreg<pair<int,int>> x("uid-x"),y("uid-y"),z("uid-z");

x.write(pair<int,int>(0,0));
y.write(pair<int,int>(1,0));
z.write(pair<int,int>(0,1));

x.join(y); x.join(z); x.resolve();

cout << x.read() << endl; // Output is ( (0,1) (1,0) )
```

Observed-Remove Map (joint design Paulo, Russell)

Embeds: CCounter, AWORSet, RWORSet, MVRegister, EWFlag, DWFlag, ORMap

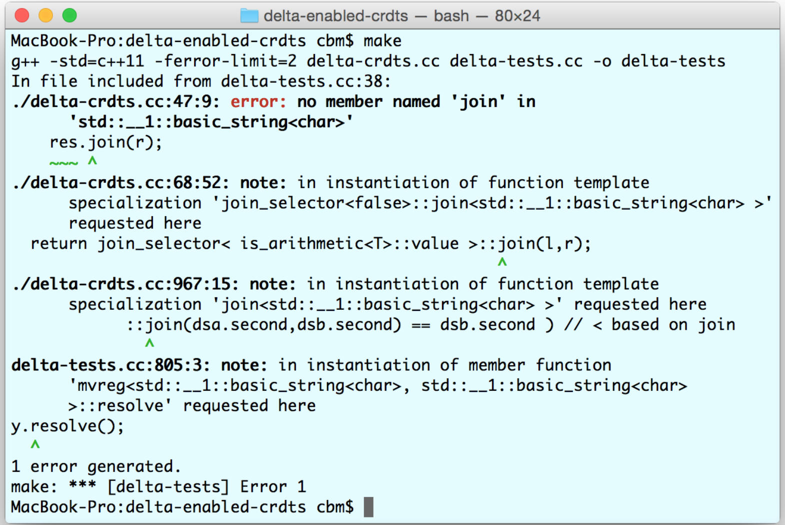
Embedded map objects share a common causal context.
Removing entries leads to a reset on the value.
One level maps

```
ormap<string,aworset<string>> mx("x"),my("y");

mx["paint"].add("blue");
my["paint"].add("red");

mx.join(my);
```

Nested maps

```
ormap<int,ormap<string,aworset<string>>> ma("alice");

ma[23]["color"].add("red␣at␣23");
ma[44]["sound"].add("loud␣at␣44");
```

C++, nice error messages at compile time



```
MacBook-Pro:delta-enabled-crdts cbm$ make
g++ -std=c++11 -ferror-limit=2 delta-crdts.cc delta-tests.cc -o delta-tests
In file included from delta-tests.cc:38:
./delta-crdts.cc:47:9: error: no member named 'join' in
      'std::__1::basic_string<char>'
    res.join(r);
    ~~~ ^
./delta-crdts.cc:68:52: note: in instantiation of function template
      specialization 'join_selector<false>::join<std::__1::basic_string<char> >'
      requested here
  return join_selector< is_arithmetic<T>::value >::join(l,r);
                                                   ^
./delta-crdts.cc:967:15: note: in instantiation of function template
      specialization 'join<std::__1::basic_string<char> >' requested here
            ::join(dsa.second,dsb.second) == dsb.second ) // < based on join
              ^
delta-tests.cc:805:3: note: in instantiation of member function
      'mvreg<std::__1::basic_string<char>, std::__1::basic_string<char>
      >::resolve' requested here
y.resolve();
  ^
1 error generated.
make: *** [delta-tests] Error 1
MacBook-Pro:delta-enabled-crdts cbm$
```

Questions?

Email: cbm@di.uminho.pt
Twitter: @xmal