Define CRDT In CRONUS

Acher Quentin

Sections of definitiion

In the folder “example”, multiple folders defining CRDTs Can be found.

To create a new CRDT Create a folder with the name of the CRDT (eg. MyCrdt)
and a go file inside with the same name “MyCrdt.go”
the go package of the file should be the same name “MyCrdt”

This Package need to include multiple definition, that we present in this pdf :
The Folder “EmptyExemple” Is showing how to implement a Operation-based CRDT (String set)
With no measurement

1) operation → Define what is an operation, and on what element they act
2) Payload definition → define what is in the payload
3) CRDT → Define the CRDT itself so we can compute the CRDT from the DAG
4) CRDTDagNode → Define the CRDTDag Node, what should be carried in every nodes,

   (Default is at least Payload, Dependency and Peer ID)

5) CRDTDag → Define the merge of noes, add of nodes, and link with Ipfs

Definition of operations

Define the Data in itself

```
type Element string
type OpNature int

const (
    OP1 OpNature = iota
    OP2
    OP3
)

type Operation struct {
    Elem Element
    Op   OpNature
}
```

Definition of Paylaods

Define what is a payload
(it only carry an operation as it is op based here, it can carry more in state/delta Based)

```
type PayloadOpBased struct {
    Op Operation
    Id string
}

func (self *PayloadOpBased) Create_PayloadOpBased(s string, o1
Operation) {

    self.Op = o1
    self.Id = s
}
```

Definition of the CRDT

Define the data of the CRDT

```
type CRDTEmptyExempleOpBased struct {
    sys     *IpfsLink.IpfsLink
    added   []string
    removed []string
}
```

Define the operations on it

```
func (self *CRDTEmptyExempleOpBased) Operation1(x string) {
    if search(self.added, x) == -1 {
        self.added = append(self.added, x)
    }
}

func (self *CRDTEmptyExempleOpBased) Operation2(x string) {…
}

func (self *CRDTEmptyExempleOpBased) Operation3(x string) {…
}
```

Lookup, to get the data's value

```
func (self *CRDTEmptyExempleOpBased) Lookup() []string {
    i := make([]string, 0)
    fmt.Println("size", len(self.added))
    for x := range self.added {
        if search(self.removed, self.added[x]) == -1 {
            i = append(i, self.added[x])
            i = append(i, ",")
        }
    }

    return i
}
```

To respect the CRDT Interface, implement a "MyCRDT.ToFile(string)" function

```
func (self *CRDTEmptyExempleOpBased) ToFile(file string) {

    b, err := json.Marshal(self)
    if err != nil {…
    }
    f, err := os.Create(file)
    if err != nil {…
    }
    f.Write(b)
    err = f.Close()
    if err != nil {…
    }
}
```

Definition of CRDTDagNode

This has to implement the CRDTDagNode Interface : “CRDTDag/CRDTDagNode”
Straight forward definition here

```
type CRDTEmptyExempleOpBasedDagNode struct {
    DagNode CRDTDag.CRDTDagNode
}

func (self *CRDTEmptyExempleOpBasedDagNode) ToFile(file string) {
    self.DagNode.ToFile(file)
}

func (self *CRDTEmptyExempleOpBasedDagNode) FromFile(fil string) {
    var pl Payload.Payload = &PayloadOpBased{}
    self.DagNode.CreateNodeFromFile(fil, &pl)
}

func (self *CRDTEmptyExempleOpBasedDagNode) GetEvent() *Payload.
Payload {
    return self.DagNode.Event
}
```

```
func (self *CRDTEmptyExempleOpBasedDagNode) GetPiD() string {
    return self.DagNode.PID
}

func (self *CRDTEmptyExempleOpBasedDagNode) GetDirect_dependency() []
CRDTDag.EncodedStr {
    return self.DagNode.DirectDependency
}


func (self *CRDTEmptyExempleOpBasedDagNode) CreateEmptyNode()
*CRDTDag.CRDTDagNodeInterface {
    n := CreateDagNode(Operation{}, "")
    var node CRDTDag.CRDTDagNodeInterface = &n
    return &node
}
```

Definition of CRDTDag

This is the biggest part of developping your CRDT in CRONUS. All the previous steps was here in order to prepare the CRDDag definition. Once the CRDTDag is well define here you can use your CRDT in CRONUS.

First use the structure CRDTDag.CRDTManager to manage your Data, and avoid lots of manual definition and file management:

```
type CRDTEmptyExempleOpBasedDag struct {
    dag CRDTDag.CRDTManager
}
```

This should implement the interface “CRDTDag/CRDTDag”, i.e. it shall implement

```
func (self *CRDTEmptyExempleOpBasedDag) Lookup_ToSpecifyType()
*CRDT.CRDT {

    crdt := CRDTEmptyExempleOpBased{…                    → Create an
    }                                                      Empty CRDT
    for x := range self.dag.GetAllNodes() {
        node := self.dag.GetAllNodesInterface()[x]       → Apply all op
        payload := (*(*node).GetEvent()).(*PayloadOpBased)   one by one,
        if payload.Op.Op == OP1 {                        → If op 1
            // fmt.Println("add")
            crdt.Operation1(string((*(*node).GetEvent()).
            (*PayloadOpBased).Op.Elem))
        } else if payload.Op.Op == OP2 {…                → If op 2
        } else {…                                        → If op 3
        }
    }
    var pl CRDT.CRDT = &crdt
    return &pl
}
```

Sending Remote update must be done in here :
CRDTDag.CRDTManager.SendRemoteUpdates()
does send CRDTDag Root nodes,
Adding measurement can be done thanks to this function

```
func (self *CRDTEmptyExempleOpBasedDag) SendRemoteUpdates() {

    self.dag.SendRemoteUpdates()
}
```

Get sys is for CRDTManager so it can access the IPFS linked to the peer

```
func (self *CRDTEmptyExempleOpBasedDag) GetSys() *IpfsLink.IpfsLink {

    return self.dag.Sys
}
```

Same for CRDTManager

```
func (self *CRDTEmptyExempleOpBasedDag) GetCRDTManager()
*CRDTDag.CRDTManager {

    return &self.dag
}
```

A more complex function detailed hereafter

```
func (self *CRDTEmptyExempleOpBasedDag) Merge
(cids []CRDTDag.EncodedStr) []string {…
}
```

More complex function in CRONUS

Four functions are complex :

1. Merge

```go
func (self *CRDTEmptyExempleOpBasedDag) Merge
(cids []CRDTDag.EncodedStr) []string {

    to_add := make([]CRDTDag.EncodedStr, 0)
    for _, cid := range cids {
        find := self.IsKnown(cid)
        if !find {
            to_add = append(to_add, cid)
        }
    }

    fils, err := self.dag.GetNodeFromEncodedCid(to_add)
    if err != nil {…
    }

    for index := range fils {
        fil := fils[index]
        // Create an Empty operation
        n := CreateDagNode(Operation{}, "")
        // Fill it with the operation just read
        n.FromFile(fil)
        // Add The node which is a Remote operation
        // It is applied like a local operation
        // But it check the dependency
        self.remoteAddNode(cids[index], n)
    }
    return fils
}
```

→ Sort out to add
only unknown Nodes

Use the function
CRDTManager.GetNodeFromEncodedCid
→ Which connect to IPFS
And retrieve all Node
With the list of Cids ‘to_add’

→ Creating a
CRDTEmptyExempleDagNode
to import the downloaded file in it.

→ Add the node with the auxiliary function
(right side)

auxiliary function :

```go
func (self *CRDTEmptyExempleOpBasedDag) remoteAddNode
(cID CRDTDag.EncodedStr, newnode CRDTEmptyExempleOpBasedDagNode) {
    var pl CRDTDag.CRDTDagNodeInterface = &newnode
    self.dag.RemoteAddNodeSuper(cID, &pl)
}
```

Calling :
CRDTManager.RemoteAddNodeSuper

Which Add the node while checking if
it knows all the dependencies.

Meaning it retrieves and add the unknown one by itself.
(following the payload type given)

More complex function in CRONUS

Four functions are complex :

2. Operations

```go
func (self *CRDTEmptyExempleOpBasedDag) OP1(x string) string {
    newNode := CreateDagNode(Operation{Elem: Element(x), Op: OP1},
    self.GetSys().IpfsNode.Identity.Pretty())
    for dependency := range self.dag.Root_nodes {
        newNode.DagNode.DirectDependency = append(newNode.DagNode.
        DirectDependency, self.dag.Root_nodes[dependency])
    }

    strFile := self.dag.NextFileName()
    if _, err := os.Stat(strFile); !errors.Is(err, os.ErrNotExist) {…
    }
    newNode.ToFile(strFile)
    bytes, err := os.ReadFile(strFile)
    if err != nil {…
    }
    path, err := self.callAddToIPFS(bytes, strFile)
    if err != nil {…
    }
    encodedCid := self.dag.EncodeCid(path)
    c := cid.Cid{}
    err = json.Unmarshal(encodedCid.Str, &c)
    if err != nil {…
    }
    var pl CRDTDag.CRDTDagNodeInterface = &newNode
    // Adding the node created before to the Merkle-DAG
    self.dag.AddNode(encodedCid, &pl)
    // Op-Based force us to send updates to other at each update
    self.SendRemoteUpdates()
    return c.String()
}
```

→ Create the dependency of the new node

→ Add the node to a file
And push it to IPFS
So the CID is known
And this new node/file
Can easily be shared to other peers

→ Add the Node with the function CRDTManager.AddNode

→ Send remote update,
As it is Op based,
this is done immediately

auxiliary function :

```go
func (self *CRDTEmptyExempleOpBasedDag) callAddToIPFS(bytes []byte,
file string) (path.Resolved, error) {
    var path path.Resolved
    var err error

    path, err = self.GetCRDTManager().AddToIPFS(self.dag.Sys, bytes)
    if err != nil {…
    }

    return path, err
}
```

Calling :
CRDTManager.AddToIPFS

Which itself connect to IPFS and return the CID of the file bytes

More complex function in CRONUS

Four functions are complex :

3. CRDT initialisation

```
func Create_CRDTSetOpBasedDag
(sys *IpfsLink.IpfsLink, cfg Config.IM_CRDTConfig)
CRDTEmptyExempleOpBasedDag {
    man := CRDTDag.Create_CRDTManager(sys, cfg.PeerName, cfg.
    BootstrapPeer, cfg.Encode, false)
    crdtSet := CRDTEmptyExempleOpBasedDag{dag: man}

    if cfg.BootstrapPeer == "" {…
    }

    var pl CRDTDag.CRDTDag = &crdtSet

    CRDTDag.CheckForRemoteUpdates(&pl, sys.Cr.Sub, man.Sys.Ctx)

    return crdtSet
}
```

→ Create the CRDTDag & Manager of the new node

→ If this peer is bootstrap Add an initial value

→ Initialize the reception of Payload Using CRDTManager.CheckForRemoteUpdate Which create a thread Checking CID receival and writing the C in a folder “./remote”

If bootstrap peer :

```
x, err := os.ReadFile("initial_value")
if err != nil {…
}
newNode := CreateDagNode(Operation{Elem: Element(x), Op:
OP1}, crdtSet.GetSys().IpfsNode.Identity.Pretty())
strFile := crdtSet.dag.NextFileName()

if _, err := os.Stat(strFile);
!errors.Is(err, os.ErrNotExist) {…
}
newNode.ToFile(strFile)
bytes, err := os.ReadFile(strFile)
if err != nil {…
}

// Add Inital State ( so it isn't counted as messages)
path, err := man.AddToIPFS(crdtSet.dag.Sys, bytes)
if err != nil {…
}

encodedCid := crdtSet.dag.EncodeCid(path)
c := cid.Cid{}
err = json.Unmarshal(encodedCid.Str, &c)
if err != nil {…
}
var pl1 CRDTDag.CRDTDagNodeInterface = &newNode

crdtSet.dag.AddNode(encodedCid, &pl1)
```

→ Add initial value Can be done outside of the rest of the experimentation Here

→ Here is an exemple of how

More complex function in CRONUS

Four functions are complex :

4. Checking received updates

```
func (self *CRDTEmptyExempleOpBasedDag) CheckUpdate(sema *semaphore.
Weighted) {
    files, err := ioutil.ReadDir(self.GetDag().
    Nodes_storage_enplacement + "/remote")
    if err != nil {…
    } else {
        to_add := make([][]byte), 0)
        for _, file := range files {
            // if the file exists, and is not currently being written
            if file.Size() > 0 && !strings.Contains(file.Name(), ".
            ArrivalTime") {
                // open the file and read it
                fil, err := os.OpenFile(self.GetDag().
                Nodes_storage_enplacement+"/remote/"+file.Name(), os.
                O RDONLY, os.ModeAppend)
                bytesread := make([]byte, stat.Size())
                n, err := fil.Read(bytesread)
                // transfer the content of the file into a CID, with
                json.Unmarshal
```

```
                if !self.IsKnown(CRDTDag.EncodedStr{Str:
                bytesread}) {
                    to_add = append(to_add, bytesread)
                }
        }
```

```
        // apply the update on the peer's data
        getSema(sema, self.GetSys().Ctx)
        self.add_cids(to_add)
        returnSema(sema)
    }
}
```