Fault Tolerant IVY Protocol (Integrated shared Virtual memory at Yale)

Go-based implementation of fault-tolerant Ivy Architecture with a backup central manager for consistent (meta)data handling during primary CM failures.

How to run the code

There are 2 types of nodes, Central Manager and Client. Each node runs on separate terminals on separate processes and communicate with each other using RPC calls. For a fresh run of the protocol, ensure the `/data` folder is empty.

Running the different types of Nodes

To start the primary CM:

1. Ensure you are in the root directory of the project.
2. Run `go build && ./ivy`
3. You will be prompted to choose the node type: "Enter Node type ('1': CM, '2': Client, 'restartCM', 'restartBackup')"
4. Type '1'. This will create a new `cm.json` file and add the primary CM object to the file.
5. The primary CM should now be running.

To start the Backup CM:

1. Run `go build && ./ivy`
2. You will be prompted to choose the node type: "Enter Node type ('1': CM, '2': Client, 'restartCM', 'restartBackup')"
3. Type '1'. This will see that a primary CM already exists in `cm.json` and add a Backup CM object to the file.
4. The Backup CM should now be running.

To start a Client:

1. Run `go build && ./ivy`
2. You will be prompted to choose the node type: "Enter Node type ('1': CM, '2': Client, 'restartCM', 'restartBackup')"
3. Type '1'. This will check `client.json` and add Client (currentHighestID + 1) to the file.
4. The Client should now be running

How to kill any Node (PrimaryCM/BackupCM/Client)

To kill any node simply go to its terminal and press `ctrl+c`

How to reboot a CM (PrimaryCM/BackupCM)

To reboot a PrimaryCM:

1. Run `go build && ./ivy`
2. You will be prompted to choose the node type: "Enter Node type ('1': CM, '2': Client, 'restartCM', 'restartBackup')"
3. Type `restartCM`. This checks `cm.json` for the IP of the original Primary CM and runs the CM on that IP.

To reboot a BackupCM:

1. Run `go build && ./ivy`
2. You will be prompted to choose the node type: "Enter Node type ('1': CM, '2': Client, 'restartCM', 'restartBackup')"
3. Type `restartBackupCM`. This checks `cm.json` for the IP of the Backup CM and runs the CM on that IP.

How to send read/write requests

- Send a write request by typing `writePage <pageNo> <content>`. For example, you type `writePage P1 Content1`.
- Send a read request by typinh `readPage <pageNo>`. For example, you type `readPage P1`.

Useful command

- CM: Type `print` to view the MetaData.
- Client: Type `print` to view the PageStore

Fault Tolerance

Syncing MetaData

When you have 2 CMs running: one Primary CM and one Backup CM, the Backup CM has runs goroutine (`go pulseCheck()`). With this, it polls the Primary CM every 1s to check if it is alive. In response, the Primary CM sends a Payload containing its MetaData. This way, every second the Backup CM has synced up with the Primary CM.

Transfer of Primary title

When the Primary CM is killed, the pulseCheck will detect the death of the Primary CM. The Backup CM now takes over as Primary and sends a `CHANGE_CM` message to all Clients in `client.json` to notify them about the change in CM. All read and write requests are now routed to the Backup CM.

Rebooting Primary CM

When you reboot the Primary CM, it sends a `IM_BACK` message to the Backup CM to let them know who's the real boss. It again sends a `CHANGE_CM` message to all the Clients to inform them about the change in CM. Read/Write requests are back to being routed to the Primary CM.

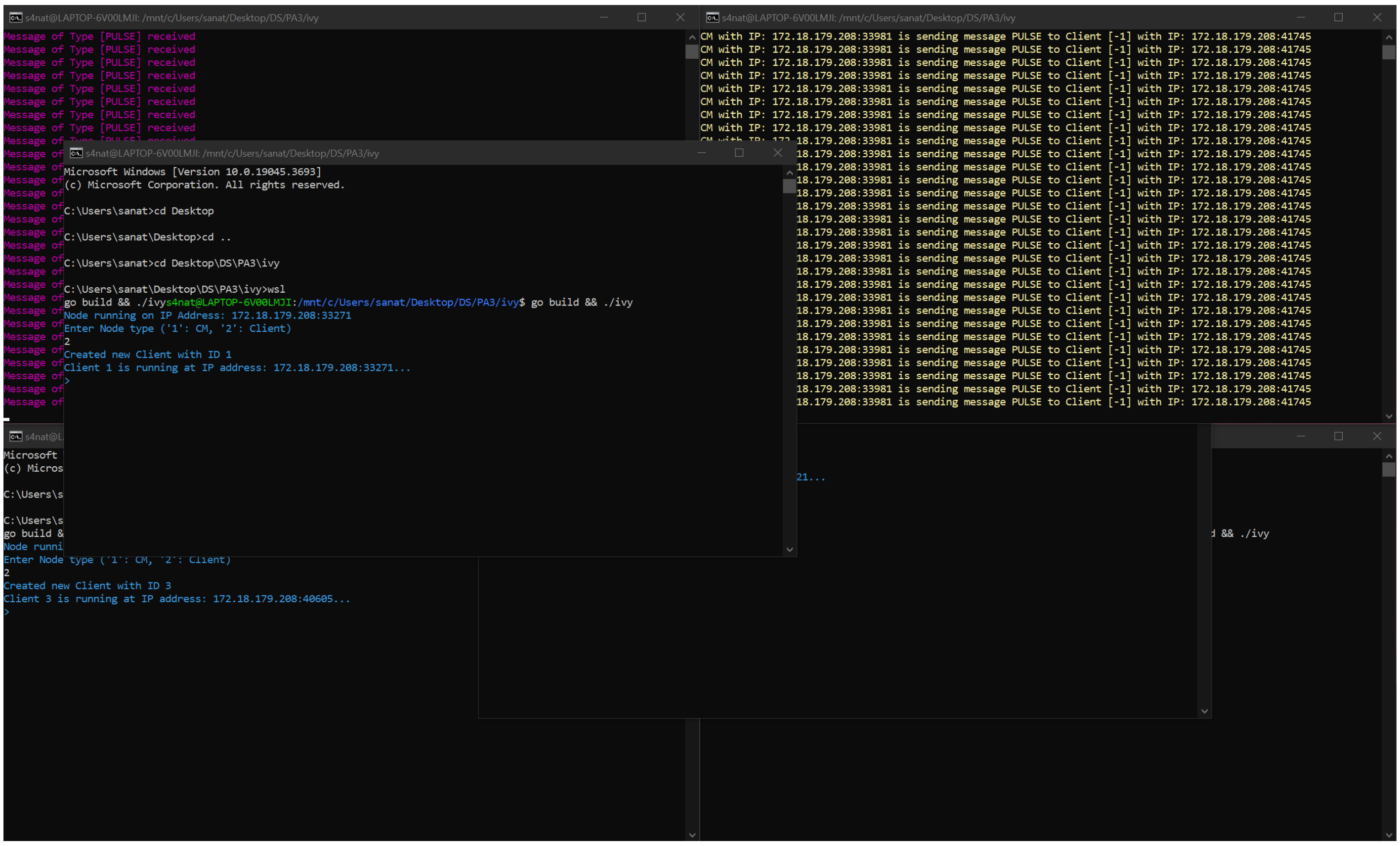
Rebooting Backup CM

In the event of Backup CM death, you can reboot it. This will simply restart the goroutine to send `PulseChecks` to the PrimaryCM and resume syncing the MetaData every 1s.

Experiments

For each test, 12 terminals were used for:

- 2 CMs (1 Primary + 1 Backup)
- 10 Clients (Client 1 - 10)
- One of the client initially seeds CM with **Pages P1-P10**
- For each scenario, all 10 Clients ran `randomizeRWRequests()` which sends a total of 10 requests randomized by a coin flip to either read or write. You can run the simulation on each Client by typing `x` .
- The time taken for all 10 requests to complete in each client was recorded (using `time.Now().UnixMilli()`).
- For each scenario the Average Time for all 10 clients to complete 10 requests is shown.



```
s4nat@LAPTOP-6V00LMJI: /mnt/c/Users/sanat/Desktop/DS/PA3/ivy
Message of Type [PULSE] received
Message of Type [PULSE] received
Message of Type [PULSE] received
Message of Type [PULSE] received
Message of Type [PULSE] received
Message of Type [PULSE] received
Message of Type [PULSE] received
Message of Type [PULSE] received
```

```
s4nat@LAPTOP-6V00LMJI: /mnt/c/Users/sanat/Desktop/DS/PA3/ivy
Microsoft Windows [Version 10.0.19045.3693]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sanat>cd Desktop

C:\Users\sanat\Desktop>cd ..

C:\Users\sanat>cd Desktop\DS\PA3\ivy

C:\Users\sanat\Desktop\DS\PA3\ivy>wsl
go build && ./ivys4nat@LAPTOP-6V00LMJI:/mnt/c/Users/sanat/Desktop/DS/PA3/ivy$ go build && ./ivy
Node running on IP Address: 172.18.179.208:33271
Enter Node type ('1': CM, '2': Client)
2
Created new Client with ID 1
Client 1 is running at IP address: 172.18.179.208:33271...
>
```

```
s4nat@LAPTOP-6V00LMJI: /mnt/c/Users/sanat/Desktop/DS/PA3/ivy
CM with IP: 172.18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
CM with IP: 172.18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
CM with IP: 172.18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
CM with IP: 172.18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
CM with IP: 172.18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
CM with IP: 172.18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
CM with IP: 172.18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
CM with IP: 172.18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
CM with IP: 172.18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
18.179.208:33981 is sending message PULSE to Client [-1] with IP: 172.18.179.208:41745
```

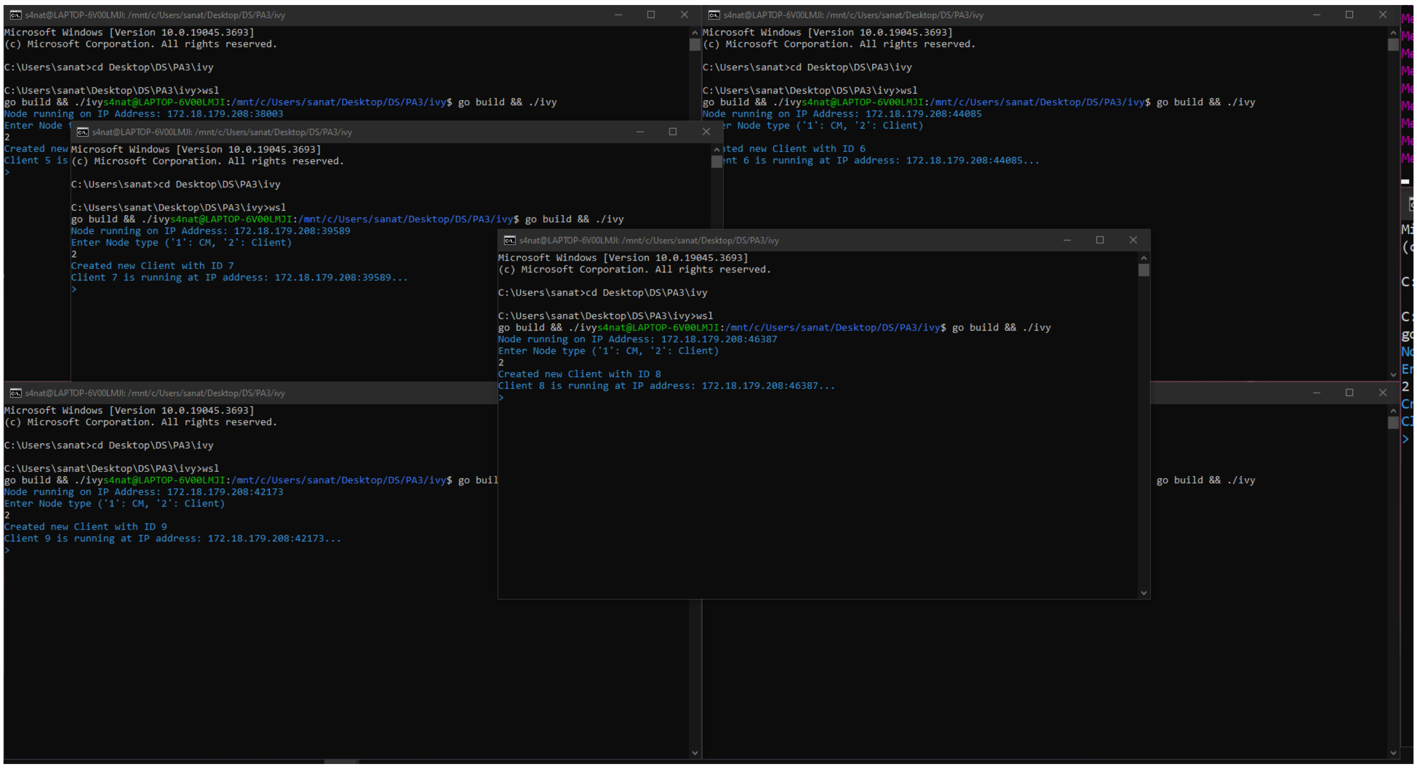
```
s4nat@L
Microsoft
(c) Micros

C:\Users\s

C:\Users\s
go build &
Node runni
Enter Node type ('1': CM, '2': Client)
2
Created new Client with ID 3
Client 3 is running at IP address: 172.18.179.208:40605...
>
```

```
21...
```

```
d && ./ivy
```



```
s4nat@LAPTOP-6V00LMJI: /mnt/c/Users/sanat/Desktop/DS/PA3/ivy
Microsoft Windows [Version 10.0.19045.3693]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sanat>cd Desktop\DS\PA3\ivy

C:\Users\sanat\Desktop\DS\PA3\ivy>wsl
go build && ./ivys4nat@LAPTOP-6V00LMJI:/mnt/c/Users/sanat/Desktop/DS/PA3/ivy$ go build && ./ivy
Node running on IP Address: 172.18.179.208:38003
Enter Node t
2
Created new
Client 5 is
>
```

```
s4nat@LAPTOP-6V00LMJI: /mnt/c/Users/sanat/Desktop/DS/PA3/ivy
Microsoft Windows [Version 10.0.19045.3693]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sanat>cd Desktop\DS\PA3\ivy

C:\Users\sanat\Desktop\DS\PA3\ivy>wsl
go build && ./ivys4nat@LAPTOP-6V00LMJI:/mnt/c/Users/sanat/Desktop/DS/PA3/ivy$ go build && ./ivy
Node running on IP Address: 172.18.179.208:44085
er Node type ('1': CM, '2': Client)

ated new Client with ID 6
ent 6 is running at IP address: 172.18.179.208:44085...
```

```
s4nat@LAPTOP-6V00LMJI: /mnt/c/Users/sanat/Desktop/DS/PA3/ivy
Microsoft Windows [Version 10.0.19045.3693]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sanat>cd Desktop\DS\PA3\ivy

C:\Users\sanat\Desktop\DS\PA3\ivy>wsl
go build && ./ivys4nat@LAPTOP-6V00LMJI:/mnt/c/Users/sanat/Desktop/DS/PA3/ivy$ go build && ./ivy
Node running on IP Address: 172.18.179.208:39589
Enter Node type ('1': CM, '2': Client)
2
Created new Client with ID 7
Client 7 is running at IP address: 172.18.179.208:39589...
>
```

```
s4nat@LAPTOP-6V00LMJI: /mnt/c/Users/sanat/Desktop/DS/PA3/ivy
Microsoft Windows [Version 10.0.19045.3693]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sanat>cd Desktop\DS\PA3\ivy

C:\Users\sanat\Desktop\DS\PA3\ivy>wsl
go build && ./ivys4nat@LAPTOP-6V00LMJI:/mnt/c/Users/sanat/Desktop/DS/PA3/ivy$ go build && ./ivy
Node running on IP Address: 172.18.179.208:46387
Enter Node type ('1': CM, '2': Client)
2
Created new Client with ID 8
Client 8 is running at IP address: 172.18.179.208:46387...
>
```

```
s4nat@LAPTOP-6V00LMJI: /mnt/c/Users/sanat/Desktop/DS/PA3/ivy
Microsoft Windows [Version 10.0.19045.3693]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sanat>cd Desktop\DS\PA3\ivy

C:\Users\sanat\Desktop\DS\PA3\ivy>wsl
go build && ./ivys4nat@LAPTOP-6V00LMJI:/mnt/c/Users/sanat/Desktop/DS/PA3/ivy$ go buil
Node running on IP Address: 172.18.179.208:42173
Enter Node type ('1': CM, '2': Client)
2
Created new Client with ID 9
Client 9 is running at IP address: 172.18.179.208:42173...
>
```

```
 go build && ./ivy
```

Test results

Scenario 1: No faults

- Type `x` on all Client terminals. No further interference.

> **Average Time/ms**: 10065

Scenario 2(a): Single Fault in Primary CM

- Type `x` on all Client terminals.
- Kill the primary CM by `ctrl-c`.

> **Average Time/ms**: 10078

Scenario 2(b): Fail and reboot Primary CM

- Type `x` on all Client terminals.
- Kill the primary CM by `ctrl-c`.
- Reboot the primary CM with `go build && ./ivy` and `restartCM`

> **Average Time/ms**: 10082**

Scenario 3: Fail and reboot Primary CM multiple times

- Type `x` on all Client terminals.
- Kill the primary CM by `ctrl-c`.
- Reboot the primary CM with `go build && ./ivy` and `restartCM`
- Repeat the killing and rebooting multiple times

> **Average Time/ms**: 10080

Scenario 4: Fail and reboot both Primary CM and Backup CM multiple times

- Type `x` on all Client terminals.
- Kill and Reboot Primary CM
  - Kill the primary CM by `ctrl-c`.
  - Reboot the primary CM with `go build && ./ivy` and `restartCM`
- Kill and Reboot Backup CM
  - Kill the Backup CM by `ctrl-c`.
  - Reboot the Backup CM with `go build && ./ivy` and `restartBackupCM`

  > **Average Time/ms**: 10082

Inferences

Due to the nature of the implementation particularly with the non blocking syncing process, the time taken for each simulation is approximately constant.

Is Fault Tolerant Version of IVY sequentially consistent?

A sequentially consistent implementation should maintain **some total ordering of read and write requests amongst all clients**. Since all requests are routed to a single Central Manager, the incoming requests to the CM are automatically ordered. In situations where proceeding to the next flow in the logic needs to be blocked while waiting for a response from all Clients, the code has checks in place to deal with it.

For example, when a CM receive a `WRITE_REQUEST`. It sends an `INVALIDATE_COPY` to all the Clients in the CopySet for the particular page. In my implementation, a `WRITE_FORWARD` is only sent when all `INVALIDATE_CONFIRMATION` are received. The code below demonstrates how the the function `return`s when a Client does not acknowledge a `INVALIDATE_COPY` message.

```
for _, clientPointer := range pageInfo.CopySet {
    invalidateCopy := Message{
      Type: INVALIDATE_COPY,
      Payload: Payload{
        InvalidateCopy: InvalidateCopy{
          WriteRequesterID: writeRequesterID,
          PageNumber:       targetPageNo,
        },
      },
    }

    reply := cm.CallRPC(invalidateCopy, CLIENT, clientPointer.ID, clientPointer.IP)
    if !reply.Ack {
      logerror.Printf("Msg [%s] from CM not acknowledged by Client %d\n", invalidateCopy.Type, clientPointer.ID)
      logerror.Println("Cannot forward Write Request")
      return
    }
  }

  // All InvalidateCopy responses have been received.
  // Send WriteForward to Page Owner
  writeForward := Message{
    Type: WRITE_FORWARD,
    Payload: Payload{
      WriteForward: WriteForward{
        WriteRequesterID: writeRequesterID,
        WriteRequesterIP: writeRequesterIP,
        PageNumber:       targetPageNo,
        Content:          content,
      },
    },
  }
  updatedPageInfo := cm.MetaData[targetPageNo]
  ownerID := updatedPageInfo.Owner.ID
  ownerIP := updatedPageInfo.Owner.IP
  reply := cm.CallRPC(writeForward, CLIENT, ownerID, ownerIP)
```

Sequential consistency with Primary CM failure

Here we have 4 cases:

1. Primary CM fails after last request is completed + Data sync with Backup CM is complete.
2. Primary CM fails before last request is completed + Data sync with Backup CM is complete.
3. Primary CM fails after last request is completed + Data sync with Backup CM is incomplete.
4. Primary CM fails before last request is completed + Data sync with Backup CM is incomplete.

Case 1

This is dealt with seamlessly as all requests are registered in MetaData and BackupCM can take over with most updated MetaData.

Case 2

In this case, the last request to the Primary CM is now lost because the the Backup CM only has MetaData complete up till the request before the last request to the Primary CM. This is not dealt with in my implementation but can easily be done. If a client does not receive a `PAGE_SEND` after sending a request, it can resend the request after a timeout.

Case 3

This is not dealt with at all. The client receives `PAGE_SEND` and assumes the request is complete. But the MetaData of the Backup CM does not reflect the latest request, hence the request is lost forever.

Case 4

This is similar to Case 2, where an incomplete request can trigger a re-request by the Client after a timeout.

Conclusion

Overall, sequential consistency is still maintained as there is no case where a request which was ordered by the CM is re-ordered in the Backup CM. That means that *some* total order of the request is always maintained. Furthermore, the likelihood of lost requests can be significantly reduced with more frequent syncing of the data with the Backup CM.