

CloudComputing

DiskVirtualization

checkPoint

Snapshot

Rollback

Submitted by: GROUP 10

Nitish Raj (2018MCS2140)

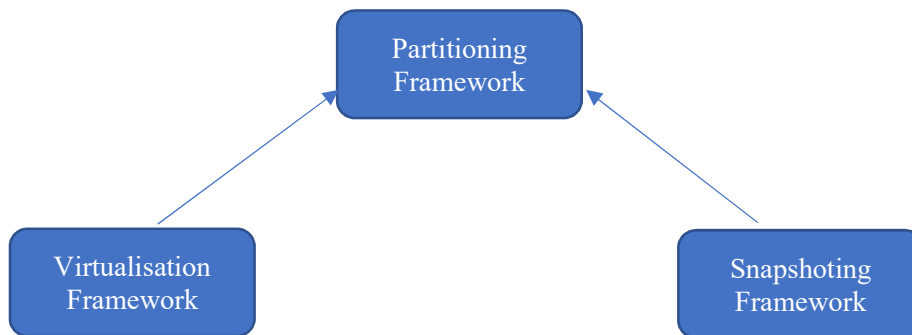
Mahendra Pratap Singh(2018MCS2120)

Prayas Sahni(2018MCS2839)

Design:

We have come up with a framework which will provide APIs for the disk virtualisation, Snapshotting, Replication.

Tree structure of framework



The PartitioningFramework.py file contains the following classes :

1. **class Block :**

This class is the basic block that contains the data and the virtual block number of its replica.

2. **class Patch :**

This class is defined so as to be able to efficiently store the patches occupied by disks, and the unused patches available in virtual space. It is stored by maintaining the starting block number and the number of blocks in the patch.

3. **class Disk :**

This class signifies a virtual disk that the APIs are used to create. It stores all the metadata about where its patches are, and also stores the information required for snapshots of the disk.

This file also instantiates the physical memory, stored as an array of objects of class Block for each physical disk.

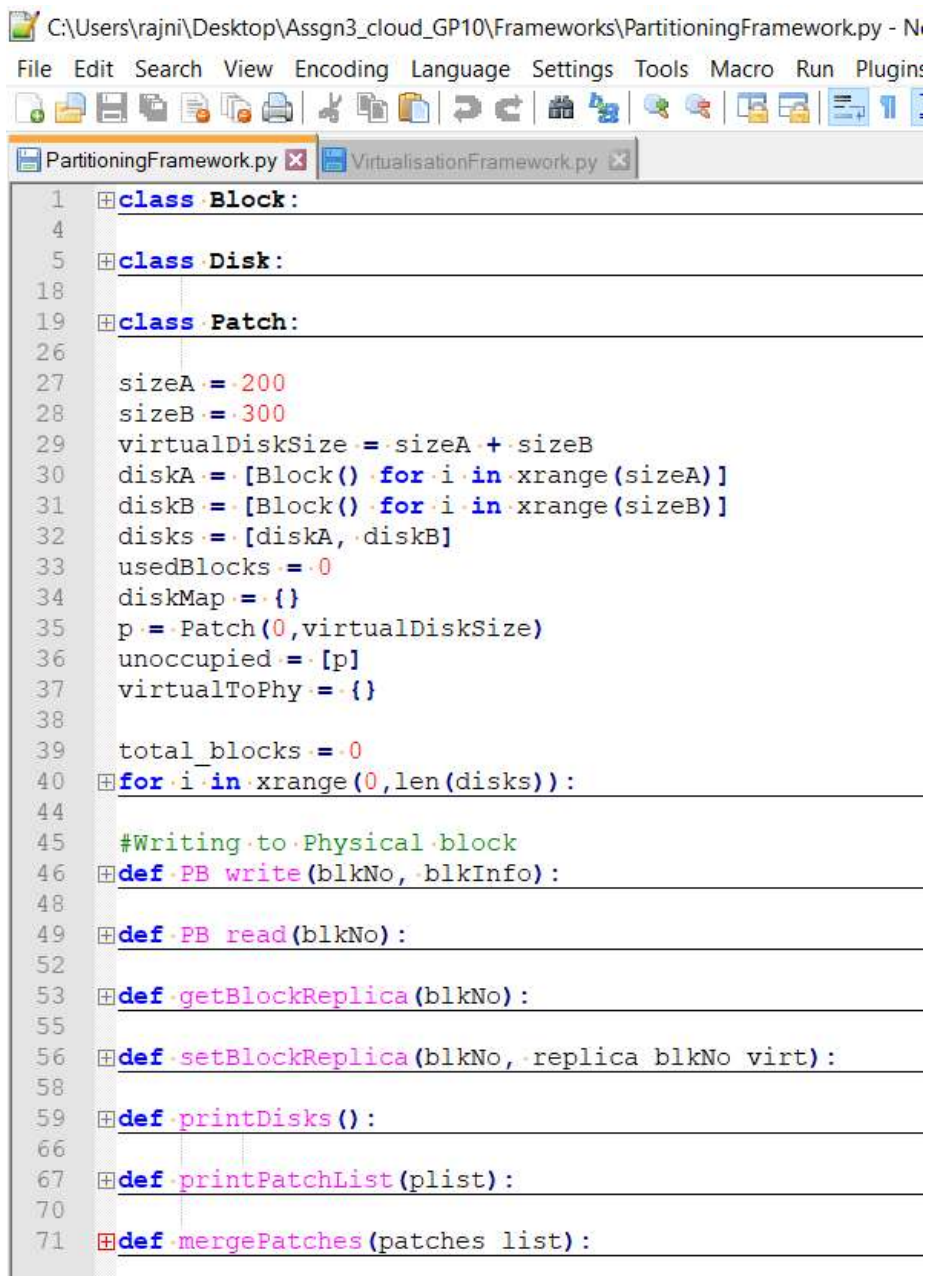
We also store the mapping from a block number in the continuous virtual space to the physical disk number and corresponding block number.

The information about unoccupied blocks is also stored as a list of patches (with blocks

indexed in virtual space)

The following methods are implemented in this file, which are internally called by the APIs exposed to the users :

- a. **PB_write()** // write physical block
- b. **PB_read()** // read physical block
- c. **getBlockReplica()**
- d. **setBlockReplica()**



```
C:\Users\rajni\Desktop\Assgn3_cloud_GP10\Frameworks\PartitioningFramework.py - N
File Edit Search View Encoding Language Settings Tools Macro Run Plugins
PartitioningFramework.py x VirtualisationFramework.py x
1  class Block:
4
5  class Disk:
18
19  class Patch:
26
27      sizeA = 200
28      sizeB = 300
29      virtualDiskSize = sizeA + sizeB
30      diskA = [Block() for i in xrange(sizeA)]
31      diskB = [Block() for i in xrange(sizeB)]
32      disks = [diskA, diskB]
33      usedBlocks = 0
34      diskMap = {}
35      p = Patch(0, virtualDiskSize)
36      unoccupied = [p]
37      virtualToPhy = {}
38
39      total_blocks = 0
40  for i in xrange(0, len(disks)):
44
45      #Writing to Physical block
46  def PB_write(blkNo, blkInfo):
48
49  def PB_read(blkNo):
52
53  def getBlockReplica(blkNo):
55
56  def setBlockReplica(blkNo, replica blkNo virt):
58
59  def printDisks():
66
67  def printPatchList(plist):
70
71  def mergePatches(patches list):
```

Implementing Disk Virtualization :

1. Consolidation & Partitioning

The virtualisation Framework (VirtualisationFramework.py) is designed to implement Disk virtualisation. Here are the details of the APIs implemented in this section :

a. **Disk_create()** -

This involves first checking if there is enough space for this disk and a disk with such an id does not exist already. Then, it calls a recursive function called Patch_create. The Patch_create function will look for the smallest unoccupied patch which can accommodate the complete disk. If no such patch exists, we need fragmentation. In that case, the largest patch is first engaged for the disk, and then Patch_create recursively looks for other unoccupied patches for the remaining portion, updating the unoccupied patches' list.

b. **DB_read()**-

Here, the main task is to convert the given block id, which is indexed relative to the particular virtual disk, to the block number in the continuous virtual space. This is done by iterating over all the patches in the disk. For this, a helper function named getVirtualDiskNo() is called.

c. **DB_write()** -

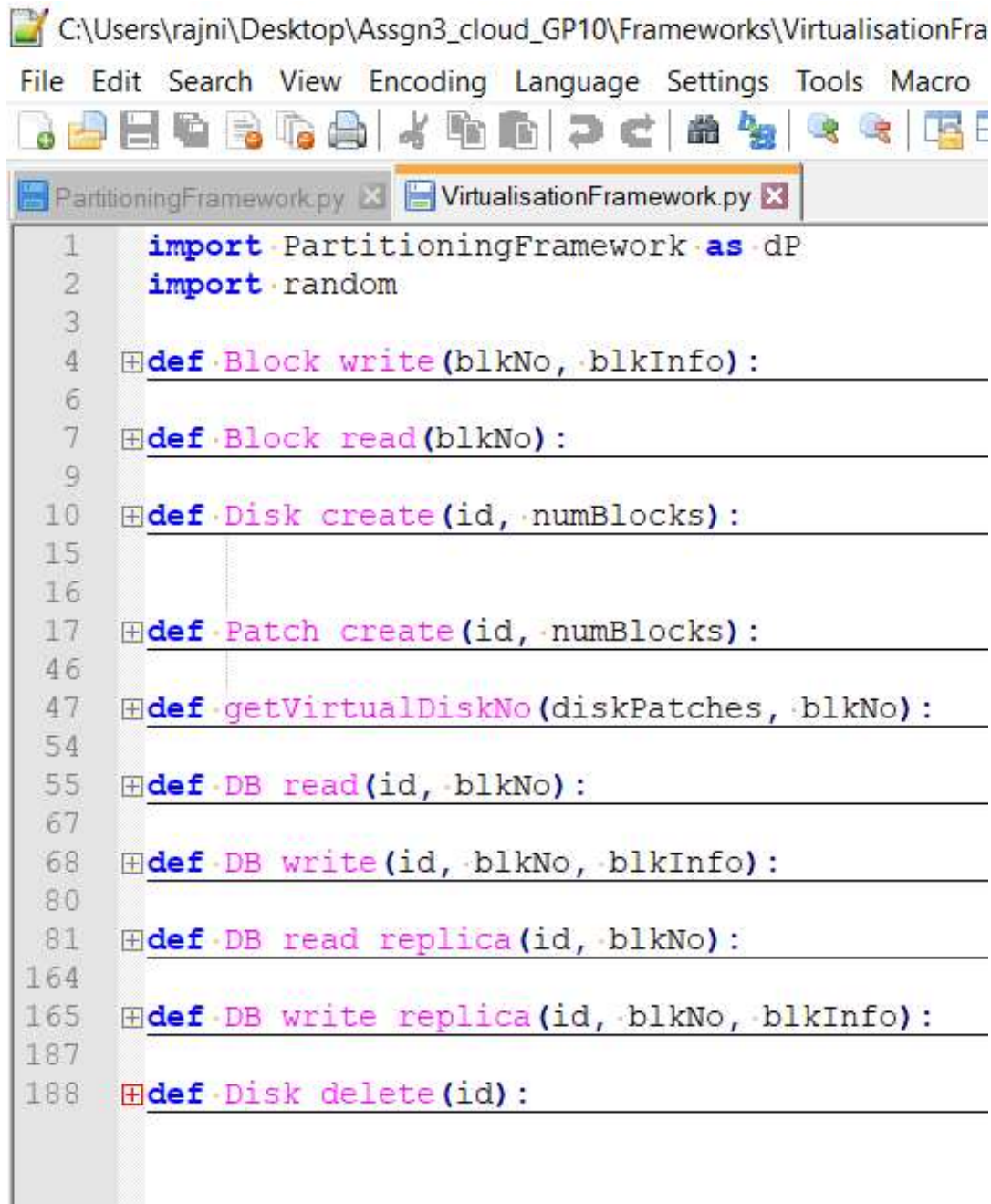
Here, similar to the reading case, we first need to call getVirtualDiskNo() function to convert the block number to an index in the virtual space. Then, it just calls the function provided by the PartitioningFramework to write data to the physical block.

d. **Disk_delete()** -

We iterate over the patches in the disk, and merge them with the list of unused patches maintained by PartitioningFramework. Also, the count of used blocks is reduced by the size of the disk. We do not delete the data actually stored in these blocks. They are considered as garbage values.

Helper functions :

- getVirtualDiskNo()



The screenshot shows a Python IDE window titled 'VirtualisationFramework.py'. The file path is 'C:\Users\rajni\Desktop\Assgn3_cloud_GP10\Frameworks\VirtualisationFra'. The menu bar includes 'File', 'Edit', 'Search', 'View', 'Encoding', 'Language', 'Settings', 'Tools', and 'Macro'. The toolbar contains various icons for file operations. The code editor shows the following Python code:

```
1  import PartitioningFramework as dP
2  import random
3
4  def Block write(blkNo, blkInfo):
5
6
7  def Block read(blkNo):
8
9
10 def Disk create(id, numBlocks):
11
12
13
14
15
16
17 def Patch create(id, numBlocks):
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47 def getVirtualDiskNo(diskPatches, blkNo):
48
49
50
51
52
53
54
55 def DB read(id, blkNo):
56
57
58
59
60
61
62
63
64
65
66
67
68 def DB write(id, blkNo, blkInfo):
69
70
71
72
73
74
75
76
77
78
79
80
81 def DB read replica(id, blkNo):
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165 def DB write replica(id, blkNo, blkInfo):
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188 def Disk delete(id):
189
```

2. Replication

Replication is implemented in the Virtualisation Framework (VirtualisationFramework.py) as well as in Snapshot Framework too (SnapshotFramework.py)

The functions in 3.1 were modified to keep a replica of all the blocks that contain data. We assume that when a disk of size x is allocated, it actually has only $x/2$ blocks, and the other half stores replicas of the first half.

The details of the implementation of the APIs are as follows :

a. **DB_read()** -

We first find a random number between 1 and 100. If it is more than 10, we simply read that block and return. If it is less than 10, it means we need to simulate a read error. Hence, we find an unallocated block to store the new replica, and then, we update the list of patches of the disk by replacing the original block by the first replica, and replacing the first replica by the new replica we just created. Note that at this point we might need to split a few patches and create a new one, due to the adjustments made. The corrupt block is removed from the disk, and also NOT put back to the list of unused patches. This means that block can NEVER be accessed again.

b. **DB_write()** -

In the write command, we need to update both the original and its replica. We first find out if the current replica block number is correct with respect to the current disk. If it's not, we set it to the virtual block number of block number + disk_size/2. This needs to be done to avoid access of any garbage value from a block. Now, we just need to write the data to both the original block and the replica block.


```
C:\Users\rajni\Desktop\Assgn3_cloud_GP10\Frameworks\SnapshotingFramework.py - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

SnapshottingFramework.py x VirtualisationFramework.py x

1 import PartitioningFramework as dP
2 import random
3
4
5 def Block write(block no, write data):
6
7
8 def Block read(block no):
9
10
11 def Disk create(id, num blocks):
12
13
14
15
16
17 def createPatch(id, num blocks):
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46 def getVirtualDiskNo(diskPatches, block no):
47
48
49
50
51
52
53
54
55 def DB read(id, block no):
56
57
58
59
60
61
62
63
64
65
66
67
68
69 def DB write(id, block no, write data):
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84 def DB read replica(id, block no):
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169 def DB write replica(id, block no, write data):
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205 def Disk delete(id):
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

VirtualisationFramework.py x

1 import PartitioningFramework as dP
2 import random
3
4 def Block write(blkNo, blkInfo):
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

def Block read(blkNo):
def Disk create(id, numBlocks):
def Patch create(id, numBlocks):
def getVirtualDiskNo(diskPatches, blkNo):
def DB read(id, blkNo):
def DB write(id, blkNo, blkInfo):
def DB read replica(id, blkNo):
def DB write replica(id, blkNo, blkInfo):
def Disk delete(id):
```

3. Snapshotting & Rolling back

This was implemented as an extension over 3.1 using Snapshotting Framework (SnapshottingFramework.py). The class Disk maintains a list of all the commands executed over this disk since creation. It also stores checkpoints as an array, where ith element of this array is the index in the list of commands which are included in this checkpoint.

Eg . 10 writes, 5 reads, checkpoint, 5 writes, checkpoint

In this case, the 1st checkpoint will store index 15, and 2nd will store index 20.

Only slight changes were done in the APIs, listed as follows :

a. **DB_write()**

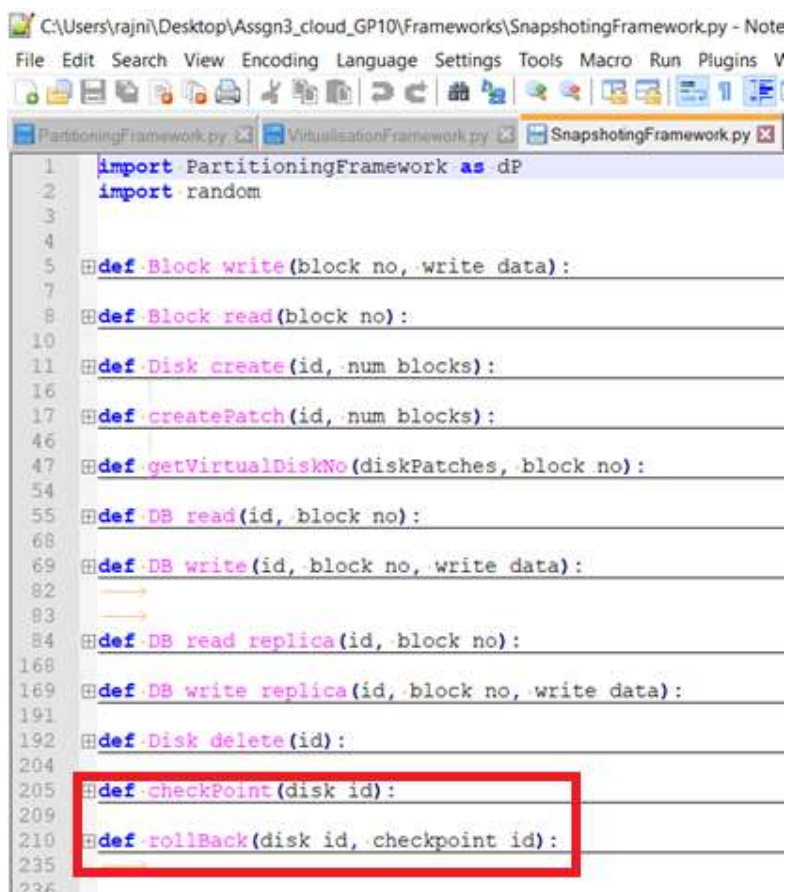
b. **DB_read()**

These APIs do one extra thing - add the corresponding operation to the list of commands. The rest of the code is the same.

The implementation of the new APIs in this section is described as follows :

- **checkpoint()** : This function simply appends the current length of commands list to the list of checkpoints. Also, it returns the index of this new check point.

- **rollback()** : This function first fetches the list of commands and checkpoints of the current disk, and then deletes it. Now, it makes a new disk, and then executes all the commands in the command list upto the checkpoint index. The new disk now stores only the list of commands it executed, and only the checkpoints that were done before the one we rolled back.



```
1 import PartitioningFramework as dP
2 import random
3
4
5 def Block write(block no, write data):
6
7
8 def Block read(block no):
9
10
11 def Disk create(id, num blocks):
12
13
14 def createPatch(id, num blocks):
15
16
17 def getVirtualDiskNo(diskPatches, block no):
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55 def DB read(id, block no):
56
57
58
59
60
61
62
63
64
65
66
67
68
69 def DB write(id, block no, write data):
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84 def DB read replica(id, block no):
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159 def DB write replica(id, block no, write data):
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192 def Disk delete(id):
193
194
195
196
197
198
199
200
201
202
203
204
205 def checkpoint(disk id):
206
207
208
209
210 def rollBack(disk id, checkpoint id):
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
```