

Міністерство освіти і науки України
НТУУ «Київський політехнічний інститут»
Фізико-технічний інститут

Протокол лабораторної роботи №1
з дисципліни Проектування високонавантажених систем
на тему: Реалізація каунтера з використанням Hazelcast

Виконав: студент групи ФІ-21
Грунда Ярослав

Київ, 2025

Мета роботи

Порівняти throughput (пропускна здатність) в залежності від структури зберігання.

Завдання

1. Реалізувати каунтер без блокувань (Distributed Map)
2. Реалізувати каунтер з використанням песимістичного блокування (Distributed Map)
3. Реалізувати каунтер з використанням оптимістичного блокування (Distributed Map)
4. Реалізувати каунтер з використанням IAtomicLong

Зміст

1	Старт роботи	2
1.1	Запуск Docker-контейнерів	2
1.2	Dockerfile	3
1.3	hazelcast.yaml	3
2	Результати	4
3	Висновки	4
4	Код	5

1 Старт роботи

1.1 Запуск Docker-контейнерів

```
# docker-compose.yml
services:
  hz-node1:
    image: hazelcast/hazelcast:5.4.0
    container_name: hz-node1
    volumes:
      - ./hazelcast.yaml:/opt/hazelcast/config/hazelcast.yaml
    environment:
      HAZELCAST_CONFIG: /opt/hazelcast/config/hazelcast.yaml
    ports:
      - "5701:5701"
    networks:
      - hazelcast-network

  hz-node2:
    image: hazelcast/hazelcast:5.4.0
    container_name: hz-node2
    volumes:
      - ./hazelcast.yaml:/opt/hazelcast/config/hazelcast.yaml
    environment:
      HAZELCAST_CONFIG: /opt/hazelcast/config/hazelcast.yaml

    ports:
      - "5702:5701"
    networks:
      - hazelcast-network

  hz-node3:
    image: hazelcast/hazelcast:5.4.0
    container_name: hz-node3
    volumes:
      - ./hazelcast.yaml:/opt/hazelcast/config/hazelcast.yaml
    environment:
      HAZELCAST_CONFIG: /opt/hazelcast/config/hazelcast.yaml
    ports:
      - "5703:5701"
    networks:
      - hazelcast-network

  management-center:
    image: hazelcast/management-center:latest
    ports:
      - "8080:8080"
    environment:
      - MC_DEFAULT_CLUSTER=lab-cluster
      - MC_DEFAULT_CLUSTER_MEMBERS=hz-node1,hz-node2,hz-node3
```

```
    networks:
      - hazelcast-network

go-client:
  build: .
  container_name: go-client
  depends_on:
    - hz-node1
    - hz-node2
    - hz-node3
  networks:
    - hazelcast-network

networks:
  hazelcast-network:
    driver: bridge
```

1.2 Dockerfile

```
# Dockerfile
FROM golang:1.25.2
WORKDIR /app
COPY . .
RUN go mod download
CMD ["go", "run", "task.go"]
```

1.3 hazelcast.yaml

```
# hazelcast.yaml
hazelcast:
  cluster-name: lab-cluster
  network:
    join:
      tcp-ip:
        enabled: true
        member-list:
          - hz-node1
          - hz-node2
          - hz-node3

cp-subsystem:
  cp-member-count: 3
  group-size: 3
```

Members

Member	Additio...	Scripting	Console	Streami...	Slow Op...	Hazelca...	Owned Partitions	Uptime	OS Committed Virtua...	OS CPU ...
hz-node1:5701	⊖	Disabled	Disabled	Enabled	No	5.4.0	91	1h 43m 31s 400ms	7.91 GB	1.81 %
hz-node2:5701	⊖	Disabled	Disabled	Enabled	No	5.4.0	90	1h 43m 31s 810ms	7.90 GB	1.81 %
hz-node3:5701	⊖	Disabled	Disabled	Enabled	No	5.4.0	90	1h 43m 32s 124ms	7.90 GB	1.81 %

1 - 3 of 3 Rows 10

CP Subsystem

✓ All CP members are accessible

CP Members Configured 3 / 3 Accessible

Promote Member to CP

Remove CP Member

Restart CP Subsystem

CP Subsystem Stats

Member	Nodes	Destroyed Groups	Active Members	Missing Members
hz-node2:5701	2	0	3	0
hz-node3:5701	2	0	3	0
hz-node1:5701	2	0	3	0

1 - 3 of 3 Rows 10

```
hz-node3 | CP Group Members [groupId: default(6622), size:3, term:1, logIndex:0] [
hz-node3 |   CPMember(uuid=0c801c9b-93d4-4726-b525-f33448346099, address=[hz-node2]:5701)
hz-node3 |   CPMember(uuid=ad0855ec-95df-4bfe-81d7-b26c40d327ca, address=[hz-node1]:5701)
hz-node3 |   CPMember(uuid=cb39dd2b-519c-4667-9328-105b0e63dff5, address=[hz-node3]:5701) - LEADER this
hz-node3 | ]
```

2 Результати

```
go-client | --- 1. Test on Distributed Map without locks ---
go-client | 2025/10/17 13:54:00 Client successfully connected to the cluster.
go-client | Expected value: 100000
go-client | Actual final value: 17371
go-client | Lost increments: 82629
go-client | Execution time: 23.0540 seconds
go-client | Throughput: 4337.64 ops/sec
go-client |
go-client | --- 2. Test with pessimistic locking ---
go-client | Expected value: 100000
go-client | Actual final value: 100000
go-client | Lost increments: 0
go-client | Execution time: 4401.2496 seconds
go-client | Throughput: 22.72 ops/sec
go-client |
go-client | --- 3. Test with optimistic locking ---
go-client | Expected value: 100000
go-client | Actual final value: 100000
go-client | Lost increments: 0
go-client | Execution time: 53.2217 seconds
go-client | Throughput: 1878.93 ops/sec
go-client |
go-client | --- 4. Test with IAtomicLong ---
go-client | Expected value: 100000
go-client | Actual final value: 100000
go-client | Lost increments: 0
go-client | Execution time: 18.2866 seconds
go-client | Throughput: 5468.47 ops/sec
```

3 Висновки

1. Підхід без блокувань (Distributed Map without locks) виявився непридатним для використання. Хоча він показав високу швидкість обробки запитів (4337 оп/с), через стан гонин-

тви (race condition) було втрачено 82.6% даних

2. Песимістичне блокування (pessimistic locking) забезпечило 100% цілісність даних, однак ціною стала катастрофічна деградація продуктивності. Пропускна здатність впала до 22.72 оп/с, а через стан гонитви (race condition) було втрачено 100% даних
3. Оптимістичне блокування (optimistic locking) стало хорошим компромісом між швидкістю та надійністю. Воно гарантувало цілісність даних і показало значно кращу продуктивність (1878.93 оп/с) порівняно з песимістичним підходом
4. Використання `IAtomcLong` продемонструвало найкращі результати. Цей підхід поєднав у собі абсолютну точність даних (0 втрат) з найвищою пропускнуою здатністю (5468.47 оп/с)

4 Код

Посилання на репозиторій: [GitHub](#)

```
1 package main
2
3 import (
4     "context"
5     "fmt"
6     "log"
7     "sync"
8     "time"
9
10    "github.com/hazelcast/hazelcast-go-client"
11 )
12
13 const (
14     NumGoroutines      = 10
15     IncrementsPerRoutine = 10000
16     ExpectedValue      = NumGoroutines * IncrementsPerRoutine
17 )
18
19 func incrementTaskNoLock(ctx context.Context, client *hazelcast.Client,
20     mapName string, wg *sync.WaitGroup) {
21     defer wg.Done()
22
23     counterMap, err := client.GetMap(ctx, mapName)
24     if err != nil {
25         log.Printf("Error GetMap: %v", err)
26         return
27     }
28
29     for i := 0; i < IncrementsPerRoutine; i++ {
30         currentValue, err := counterMap.Get(ctx, "counter")
31         if err != nil {
32             log.Printf("Error Get: %v", err)
33             continue
34         }
35         val, _ := currentValue.(int32)
```

```
35         _, err = counterMap.Put(ctx, "counter", val+1)
36         if err != nil {
37             log.Printf("Error Put: %v", err)
38         }
39     }
40 }
41
42 func incrementTaskWithPessimisticLock(ctx context.Context, client *
43     hazelcast.Client, mapName string, wg *sync.WaitGroup) {
44     defer wg.Done()
45
46     counterMap, err := client.GetMap(ctx, mapName)
47     if err != nil {
48         log.Printf("Error GetMap: %v", err)
49         return
50     }
51
52     lockCtx := counterMap.NewLockContext(ctx)
53
54     for i := 0; i < IncrementsPerRoutine; i++ {
55         if err := counterMap.Lock(lockCtx, "counter"); err != nil
56         {
57             log.Printf("Error Lock: %v", err)
58             continue
59         }
60
61         currentValue, err := counterMap.Get(lockCtx, "counter")
62         if err != nil {
63             log.Printf("Error Get inside lock: %v", err)
64             _ = counterMap.Unlock(lockCtx, "counter")
65             continue
66         }
67
68         val, ok := currentValue.(int32)
69         if !ok {
70             log.Printf("Failed to cast value to int32, got: %
71                 T", currentValue)
72             _ = counterMap.Unlock(lockCtx, "counter")
73             continue
74         }
75
76         _, err = counterMap.Put(lockCtx, "counter", val+1)
77         if err != nil {
78             log.Printf("Error Put inside lock: %v", err)
79             _ = counterMap.Unlock(lockCtx, "counter")
80             continue
81         }
82
83         if err := counterMap.Unlock(lockCtx, "counter"); err !=
84             nil {
85             log.Printf("Error Unlock: %v", err)
86         }
87     }
```

```
84 }
85
86 func incrementTaskWithOptimisticLock(ctx context.Context, client *
    hazelcast.Client, mapName string, wg *sync.WaitGroup) {
87     defer wg.Done()
88
89     counterMap, err := client.GetMap(ctx, mapName)
90     if err != nil {
91         log.Printf("Error GetMap: %v", err)
92         return
93     }
94
95     for i := 0; i < IncrementsPerRoutine; i++ {
96         for {
97             oldValue, err := counterMap.Get(ctx, "counter")
98             if err != nil {
99                 log.Printf("Error Get: %v", err)
100                 time.Sleep(10 * time.Millisecond)
101                 continue
102             }
103             val, _ := oldValue.(int32)
104
105             newValue := val + 1
106
107             replaced, err := counterMap.ReplaceIfSame(ctx, "
                counter", oldValue, newValue)
108             if err != nil {
109                 log.Printf("Error ReplaceIfSame: %v", err
                    )
110                 time.Sleep(10 * time.Millisecond)
111                 continue
112             }
113
114             if replaced {
115                 break
116             }
117         }
118     }
119 }
120
121 func incrementTaskWithIAAtomicLong(ctx context.Context, atomicLong *
    hazelcast.AtomicLong, wg *sync.WaitGroup) {
122     defer wg.Done()
123     for i := 0; i < IncrementsPerRoutine; i++ {
124         if _, err := atomicLong.IncrementAndGet(ctx); err != nil
            {
125             log.Printf("Error IncrementAndGet: %v", err)
126         }
127     }
128 }
129
130 func main() {
131     ctx := context.Background()
```



```

132     config := hazelcast.NewConfig()
133     config.Cluster.Name = "lab-cluster"
134     config.Cluster.Network.SetAddresses("hz-node1:5701", "hz-node2
        :5701", "hz-node3:5701")
135
136     time.Sleep(5 * time.Second)
137
138     client, err := hazelcast.StartNewClientWithConfig(ctx, config)
139     if err != nil {
140         log.Fatalf("Failed to start Hazelcast client: %v", err)
141     }
142     defer client.Shutdown(ctx)
143     log.Println("Client successfully connected to the cluster.")
144
145     runMapTest(ctx, client, "1. Test on Distributed Map without locks
        ", "map_no_lock", incrementTaskNoLock)
146     runMapTest(ctx, client, "2. Test with pessimistic locking", "
        map_pessimistic", incrementTaskWithPessimisticLock)
147     runMapTest(ctx, client, "3. Test with optimistic locking", "
        map_optimistic", incrementTaskWithOptimisticLock)
148     runAtomicLongTest(ctx, client)
149 }
150
151 func runMapTest(ctx context.Context, client *hazelcast.Client, title,
    mapName string, task func(context.Context, *hazelcast.Client, string,
    *sync.WaitGroup)) {
152     fmt.Printf("\n--- %s ---\n", title)
153     counterMap, err := client.GetMap(ctx, mapName)
154     if err != nil {
155         log.Printf("Error getting map %s: %v", mapName, err)
156         return
157     }
158     counterMap.Put(ctx, "counter", int32(0))
159
160     var wg sync.WaitGroup
161     startTime := time.Now()
162     for i := 0; i < NumGoroutines; i++ {
163         wg.Add(1)
164         go task(ctx, client, mapName, &wg)
165     }
166     wg.Wait()
167     printResults(ctx, time.Since(startTime), counterMap, nil)
168 }
169
170 func runAtomicLongTest(ctx context.Context, client *hazelcast.Client) {
171     fmt.Println("\n--- 4. Test with IAtomicLong ---")
172     atomicCounter, err := client.CPSubsystem().GetAtomicLong(ctx, "
        atomic_long_counter")
173     if err != nil {
174         log.Printf("Error getting AtomicLong: %v", err)
175         return
176     }
177     atomicCounter.Set(ctx, 0)

```

```

178
179     var wg sync.WaitGroup
180     startTime := time.Now()
181     for i := 0; i < NumGoroutines; i++ {
182         wg.Add(1)
183         go incrementTaskWithIAAtomicLong(ctx, atomicCounter, &wg)
184     }
185     wg.Wait()
186     printResults(ctx, time.Since(startTime), nil, atomicCounter)
187 }
188
189 func printResults(ctx context.Context, duration time.Duration, m *
    hazelcast.Map, al *hazelcast.AtomicLong) {
190     var finalValue interface{}
191     var err error
192
193     if m != nil {
194         finalValue, err = m.Get(ctx, "counter")
195     } else if al != nil {
196         finalValue, err = al.Get(ctx)
197     }
198
199     if err != nil {
200         log.Printf("Failed to get final value: %v", err)
201         return
202     }
203
204     fmt.Printf("Expected value: %d\n", ExpectedValue)
205     fmt.Printf("Actual final value: %v\n", finalValue)
206
207     var lost int64
208     switch v := finalValue.(type) {
209     case int32:
210         lost = int64(ExpectedValue) - int64(v)
211     case int64:
212         lost = int64(ExpectedValue) - v
213     }
214
215     totalOps := float64(NumGoroutines * IncrementsPerRoutine)
216     throughput := totalOps / duration.Seconds()
217
218     fmt.Printf("Lost increments: %d\n", lost)
219     fmt.Printf("Execution time: %.4f seconds\n", duration.Seconds())
220     fmt.Printf("Throughput: %.2f ops/sec\n", throughput)
221 }

```