HW5 Report, Chordy: A Distributed Hash Table

David Fischer

October 5, 2025

1 Introduction

This assignment's goal was the implementation of *Chordy* a DHT based on a variation of *Chord*, a distributed lookup protocol designed to efficiently identify the node in a peer-to-peer that stores a particular data item. *Chord* addresses many problems distributed systems face such as load balancing, decentralization, scalability, availability, and flexible naming.

2 Implementation

2.1 Ring Structure

To ease validation and development, instead of the usually intended consistent hashing function, simple numerical IDs were used for both nodes and later on keys. This still provides the random distribution of keys to nodes, which should result in even storage sizes between nodes. As opposed to the *finger table*, used in the original *Chord* paper, which allows for $O(logN)$ lookup times, a simpler routing table was implemented. Each node only keeps track of its successor and predecessor in the ring. This results in $O(N)$ lookup times. The ring structure is periodically stabilized through a `stabilize/3` function, which gradually heals the ring when new nodes join by having each node ask its successor for their predecessor and adjusting its own records accordingly. As the structure will have to wrap around at some point, the `between/3` function was implemented in a way to handle `From` being larger than `To` by checking if the `Id` is either larger than `From` or smaller than `To`.

2.2 Storage

Storage was implemented by having each node keep a map of key-value pairs which can be added and looked up by sending corresponding messages to the nodes. These messages are routed through the ring structure until they reach the node responsible for the given key, at which point the operation is executed and the client is notified through a message including the reference it provided using make_ref/0. Additionally, to support new nodes joining the ring, a handover message containing the keys split from the predecessor, based on the two nodes keys, is sent to the new node.

2.3 Failure Detection

To detect failed nodes, Erlangs failure detection mechanism was used to have each node monitor their predecessor and successor. Nodes keep a record of their successors successor, which is provided during each stabilization round. This is done to maintain structure should their original successor fail. Should the predecessor fail, nodes set it to nil and wait for the predecessors predecessor to notify them. This approach is however not ideal if two neighboring nodes fail concurrently.

2.4 Data Replication

As data is only stored in memory, node failures result in data loss. To mitigate this, nodes keep a replica of their predecessors data which is built concurrently with the main storage. The client is only notified of a successful add operation once both the main storage and the replica have been updated. Both the handover and failure handlers were updated to correctly split and merge the replica storage depending on the state of the ring. In simple terms, when a predecessor dies, the node merges the replica storage with its main storage and propagates its new state to its successor. When a new node joins, it receives the relevant keys for both its main storage and replica storage from its predecessor.

3 Main problems and solutions

3.1 Functions and Parameters

The amount of functions and parameters the mature implementations require posed difficulties with retaining a holistic view and often lead to failing tests due to return value mismatches and bugs. This was resolved through meticulous analysis and stepping through the node implementations function for function.

3.2 Replication

Implementing functional data replication posed the greatest challenge as it required considerations in every part of the system. A rudimentary early implementation used the synchronization cycles to periodically correct the replicas, however this was inefficient and would strain the network. Stepping through the implementation scenario for scenario helped identify the necessary changes and a corresponding sketch was created in parallel to aid development, see Figure 2.

4 Evaluation

4.1 Performance

Figure 1 shows the performance of the DHT with different ring sizes and amounts of clients, each of the client machines performing 1000 operations over the first node in the ring. The heatmaps illustrate the routing tables $O(N)$ lookup times, as performance degrades more with the number of nodes in the ring than with the number of clients. The single entrypoint into the ring is also bottleneck which could be resolved by having clients randomly hit nodes to profit from the pseudo-random key distribution.

4.2 Ring Maintenance

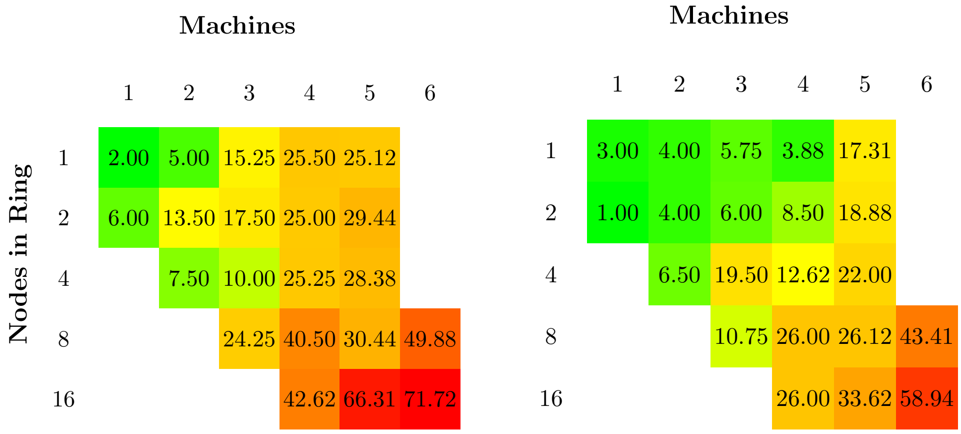
When a ring is stabilized on a short delay, the convergence is sped up and thus lookup and add operations get to the correct node faster. Key responsibility and handovers are also sped up. This however causes the system to use more resources both in terms of compute and network usage, specifically nodes generating messages for *request*, *notify*, and *status* each round.

5 Conclusions

The advantages of this distributed storage come both in performance, through load balancing, and fault tolerance, through self healing and the lack of a single point of failure. Fault tolerance should be however prioritized, especially in productive applications, as data loss is often not acceptable.

Future improvements to the system include routing table improvements with a solution as the *finger table* to improve lookup timings and considering network latency, although this would significantly increase complexity. A better replica mechanism which is either periodically synchronized through the stabilize function, or persisted on disk which nodes automatically restarting and recovering their state. Replicas could also be used to improve read performance, although this requires dealing with consistency issues. Finally, either an eventual consistency model could be applied to improve availability, or two-phase commits to ensure consistency, both excluding the other due to the CAP theorem.

A Performance Heatmaps



	Machines					
Nodes in Ring	1	2	3	4	5	6
1	2.00	5.00	15.25	25.50	25.12	
2	6.00	13.50	17.50	25.00	29.44	
4		7.50	10.00	25.25	28.38	
8			24.25	40.50	30.44	49.88
16				42.62	66.31	71.72

(a) Add Operations

	Machines					
	1	2	3	4	5	6
1	3.00	4.00	5.75	3.88	17.31	
2	1.00	4.00	6.00	8.50	18.88	
4		6.50	19.50	12.62	22.00	
8			10.75	26.00	26.12	43.41
16				26.00	33.62	58.94

(b) Lookup Operations

Figure 1: Performance comparison in milliseconds with varying numbers of nodes and clients.

B Storage Replication Sketch

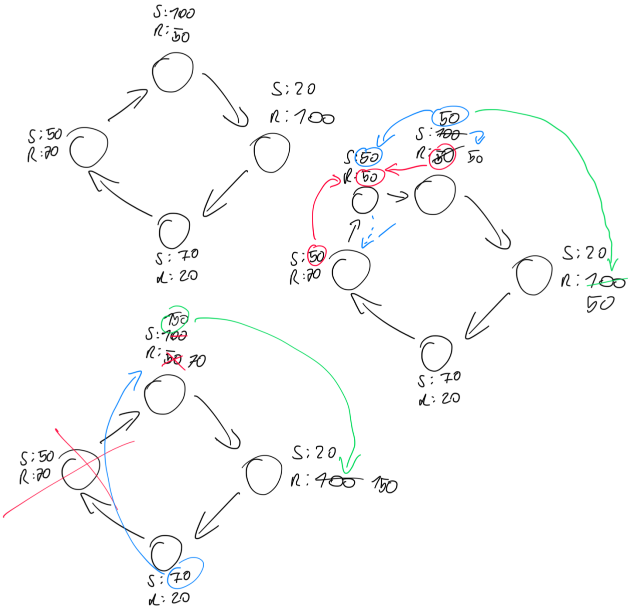
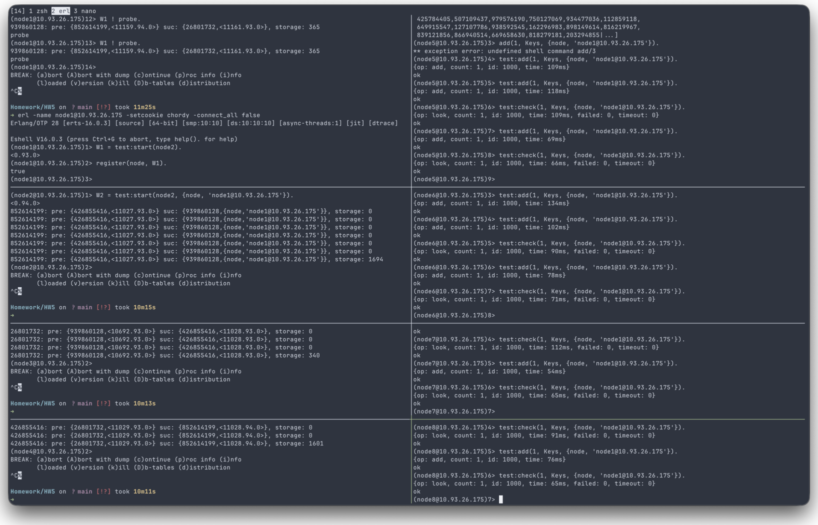


Figure 2: Sketch of the replication mechanism.

C Testing



```
[14] 1 zsh 2 erl 3 nano
(node1@10.93.26.175)12> W1 ! probe.
939860128: pre: {852614199,<11159.94.0>} suc: {26801732,<11161.93.0>}, storage: 365
probe
(node1@10.93.26.175)13> W1 ! probe.
939860128: pre: {852614199,<11159.94.0>} suc: {26801732,<11161.93.0>}, storage: 365
probe
(node1@10.93.26.175)14>
BREAK: (a)bort (A)bort with dump (c)ontinue (p)roc info (i)nfo
       (l)oaded (v)ersion (k)ill (D)b-tables (d)istribution
^C%

Homework/HW5 on  main [!?] took 11m25s
→ erl -name node1@10.93.26.175 -setcookie chordy -connect_all false
Erlang/OTP 28 [erts-16.0.3] [source] [64-bit] [smp:10:10] [ds:10:10:10] [async-threads:1] [jit] [dtrace]

Eshell V16.0.3 (press Ctrl+G to abort, type help(). for help)
(node1@10.93.26.175)1> W1 = test:start(node2).
<0.93.0>
(node1@10.93.26.175)2> register(node, W1).
true
(node1@10.93.26.175)3>
```

```
(node2@10.93.26.175)1> W2 = test:start(node2, {node, 'node1@10.93.26.175'}).
<0.94.0>
852614199: pre: {426855416,<11027.93.0>} suc: {939860128,{node,'node1@10.93.26.175'}}, storage: 0
852614199: pre: {426855416,<11027.93.0>} suc: {939860128,{node,'node1@10.93.26.175'}}, storage: 0
852614199: pre: {426855416,<11027.93.0>} suc: {939860128,{node,'node1@10.93.26.175'}}, storage: 0
852614199: pre: {426855416,<11027.93.0>} suc: {939860128,{node,'node1@10.93.26.175'}}, storage: 0
852614199: pre: {426855416,<11027.93.0>} suc: {939860128,{node,'node1@10.93.26.175'}}, storage: 0
852614199: pre: {426855416,<11027.93.0>} suc: {939860128,{node,'node1@10.93.26.175'}}, storage: 0
852614199: pre: {426855416,<11027.93.0>} suc: {939860128,{node,'node1@10.93.26.175'}}, storage: 1694
(node2@10.93.26.175)2>
BREAK: (a)bort (A)bort with dump (c)ontinue (p)roc info (i)nfo
       (l)oaded (v)ersion (k)ill (D)b-tables (d)istribution
^C%

Homework/HW5 on  main [!?] took 10m15s
→
```

```
26801732: pre: {939860128,<10692.93.0>} suc: {426855416,<11028.93.0>}, storage: 0
26801732: pre: {939860128,<10692.93.0>} suc: {426855416,<11028.93.0>}, storage: 0
26801732: pre: {939860128,<10692.93.0>} suc: {426855416,<11028.93.0>}, storage: 0
26801732: pre: {939860128,<10692.93.0>} suc: {426855416,<11028.93.0>}, storage: 340
(node3@10.93.26.175)2>
BREAK: (a)bort (A)bort with dump (c)ontinue (p)roc info (i)nfo
       (l)oaded (v)ersion (k)ill (D)b-tables (d)istribution
^C%

Homework/HW5 on  main [!?] took 10m13s
→
```

```
426855416: pre: {26801732,<11029.93.0>} suc: {852614199,<11028.94.0>}, storage: 0
426855416: pre: {26801732,<11029.93.0>} suc: {852614199,<11028.94.0>}, storage: 0
426855416: pre: {26801732,<11029.93.0>} suc: {852614199,<11028.94.0>}, storage: 1601
(node4@10.93.26.175)2>
BREAK: (a)bort (A)bort with dump (c)ontinue (p)roc info (i)nfo
       (l)oaded (v)ersion (k)ill (D)b-tables (d)istribution
^C%

Homework/HW5 on  main [!?] took 10m11s
→
```

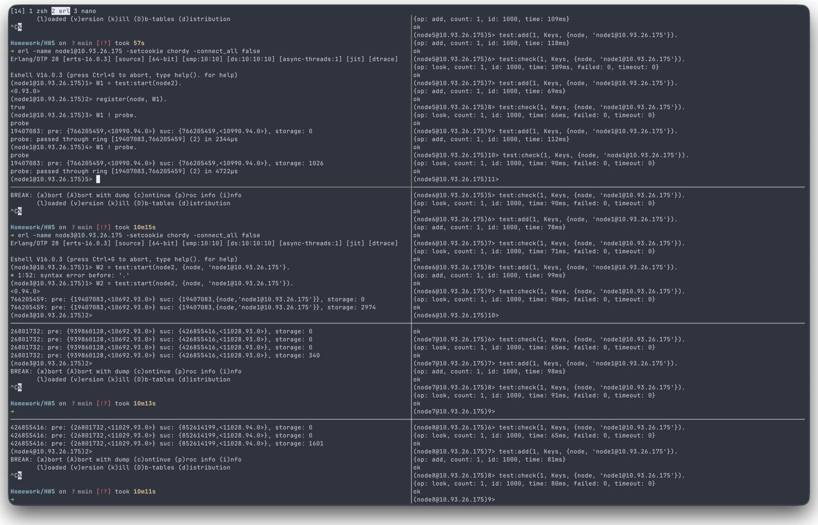
```
 425784405,507109437,979576190,750127069,934477036,112859118,
 649915547,127107786,938592545,162296983,898149614,816219967,
 839121856,866940514,669658630,818279181,203294855|...]
(node5@10.93.26.175)3> add(1, Keys, {node, 'node1@10.93.26.175'}).
** exception error: undefined shell command add/3
(node5@10.93.26.175)4> test:add(1, Keys, {node, 'node1@10.93.26.175'}).
{op: add, count: 1, id: 1000, time: 109ms}
ok
(node5@10.93.26.175)5> test:add(1, Keys, {node, 'node1@10.93.26.175'}).
{op: add, count: 1, id: 1000, time: 118ms}
ok
(node5@10.93.26.175)6> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 109ms, failed: 0, timeout: 0}
ok
(node5@10.93.26.175)7> test:add(1, Keys, {node, 'node1@10.93.26.175'}).
{op: add, count: 1, id: 1000, time: 69ms}
ok
(node5@10.93.26.175)8> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 66ms, failed: 0, timeout: 0}
ok
(node5@10.93.26.175)9>
```

```
(node6@10.93.26.175)3> test:add(1, Keys, {node, 'node1@10.93.26.175'}).
{op: add, count: 1, id: 1000, time: 134ms}
ok
(node6@10.93.26.175)4> test:add(1, Keys, {node, 'node1@10.93.26.175'}).
{op: add, count: 1, id: 1000, time: 102ms}
ok
(node6@10.93.26.175)5> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 90ms, failed: 0, timeout: 0}
ok
(node6@10.93.26.175)6> test:add(1, Keys, {node, 'node1@10.93.26.175'}).
{op: add, count: 1, id: 1000, time: 78ms}
ok
(node6@10.93.26.175)7> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 71ms, failed: 0, timeout: 0}
ok
(node6@10.93.26.175)8>
```

```
ok
(node7@10.93.26.175)4> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 112ms, failed: 0, timeout: 0}
ok
(node7@10.93.26.175)5> test:add(1, Keys, {node, 'node1@10.93.26.175'}).
{op: add, count: 1, id: 1000, time: 54ms}
ok
(node7@10.93.26.175)6> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 65ms, failed: 0, timeout: 0}
ok
(node7@10.93.26.175)7>
```

```
(node8@10.93.26.175)4> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 91ms, failed: 0, timeout: 0}
ok
(node8@10.93.26.175)5> test:add(1, Keys, {node, 'node1@10.93.26.175'}).
{op: add, count: 1, id: 1000, time: 76ms}
ok
(node8@10.93.26.175)6> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 65ms, failed: 0, timeout: 0}
ok
(node8@10.93.26.175)7>
```

Figure 3: Single node ring test.



```
[14] 1 zsh 2 erl 3 nano
       (l)oaded (v)ersion (k)ill (D)b-tables (d)istribution
^C%

Homework/HW5 on  main [!?] took 57s
→ erl -name node1@10.93.26.175 -setcookie chordy -connect_all false
Erlang/OTP 28 [erts-16.0.3] [source] [64-bit] [smp:10:10] [ds:10:10:10] [async-threads:1] [jit] [dtrace]

Eshell V16.0.3 (press Ctrl+G to abort, type help(). for help)
(node1@10.93.26.175)1> W1 = test:start(node2).
<0.93.0>
(node1@10.93.26.175)2> register(node, W1).
true
(node1@10.93.26.175)3> W1 ! probe.
probe
19407083: pre: {766205459,<10990.94.0>} suc: {766205459,<10990.94.0>}, storage: 0
probe: passed through ring [19407083,766205459] (2) in 2344μs
(node1@10.93.26.175)4> W1 ! probe.
probe
19407083: pre: {766205459,<10990.94.0>} suc: {766205459,<10990.94.0>}, storage: 1026
probe: passed through ring [19407083,766205459] (2) in 4722μs
(node1@10.93.26.175)5>
```

```
BREAK: (a)bort (A)bort with dump (c)ontinue (p)roc info (i)nfo
       (l)oaded (v)ersion (k)ill (D)b-tables (d)istribution
^C%

Homework/HW5 on  main [!?] took 10m15s
→ erl -name node3@10.93.26.175 -setcookie chordy -connect_all false
Erlang/OTP 28 [erts-16.0.3] [source] [64-bit] [smp:10:10] [ds:10:10:10] [async-threads:1] [jit] [dtrace]

Eshell V16.0.3 (press Ctrl+G to abort, type help(). for help)
(node3@10.93.26.175)1> W2 = test:start(node2, {node, 'node1@10.93.26.175'}.
* 1:52: syntax error before: '.'
(node3@10.93.26.175)1> W2 = test:start(node2, {node, 'node1@10.93.26.175'}).
<0.94.0>
766205459: pre: {19407083,<10692.93.0>} suc: {19407083,{node,'node1@10.93.26.175'}}, storage: 0
766205459: pre: {19407083,<10692.93.0>} suc: {19407083,{node,'node1@10.93.26.175'}}, storage: 2974
(node3@10.93.26.175)2>
```

```
26801732: pre: {939860128,<10692.93.0>} suc: {426855416,<11028.93.0>}, storage: 0
26801732: pre: {939860128,<10692.93.0>} suc: {426855416,<11028.93.0>}, storage: 0
26801732: pre: {939860128,<10692.93.0>} suc: {426855416,<11028.93.0>}, storage: 0
26801732: pre: {939860128,<10692.93.0>} suc: {426855416,<11028.93.0>}, storage: 340
(node3@10.93.26.175)2>
BREAK: (a)bort (A)bort with dump (c)ontinue (p)roc info (i)nfo
       (l)oaded (v)ersion (k)ill (D)b-tables (d)istribution
^C%

Homework/HW5 on  main [!?] took 10m13s
→
```

```
426855416: pre: {26801732,<11029.93.0>} suc: {852614199,<11028.94.0>}, storage: 0
426855416: pre: {26801732,<11029.93.0>} suc: {852614199,<11028.94.0>}, storage: 0
426855416: pre: {26801732,<11029.93.0>} suc: {852614199,<11028.94.0>}, storage: 1601
(node4@10.93.26.175)2>
BREAK: (a)bort (A)bort with dump (c)ontinue (p)roc info (i)nfo
       (l)oaded (v)ersion (k)ill (D)b-tables (d)istribution
^C%

Homework/HW5 on  main [!?] took 10m11s
→
```

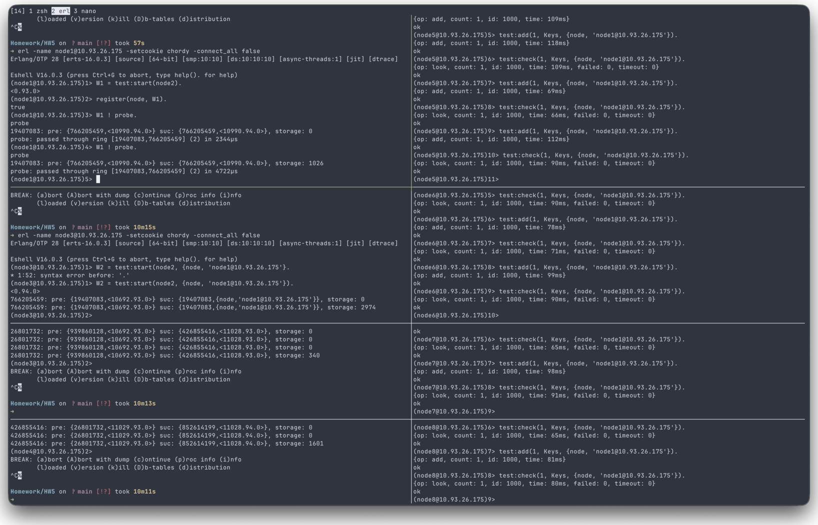
```
{op: add, count: 1, id: 1000, time: 109ms}
ok
(node5@10.93.26.175)5> test:add(1, Keys, {node, 'node1@10.93.26.175'}).
{op: add, count: 1, id: 1000, time: 118ms}
ok
(node5@10.93.26.175)6> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 109ms, failed: 0, timeout: 0}
ok
(node5@10.93.26.175)7> test:add(1, Keys, {node, 'node1@10.93.26.175'}).
{op: add, count: 1, id: 1000, time: 69ms}
ok
(node5@10.93.26.175)8> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 66ms, failed: 0, timeout: 0}
ok
(node5@10.93.26.175)9> test:add(1, Keys, {node, 'node1@10.93.26.175'}).
{op: add, count: 1, id: 1000, time: 112ms}
ok
(node5@10.93.26.175)10> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 90ms, failed: 0, timeout: 0}
ok
(node5@10.93.26.175)11>
```

```
(node6@10.93.26.175)5> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 90ms, failed: 0, timeout: 0}
ok
(node6@10.93.26.175)6> test:add(1, Keys, {node, 'node1@10.93.26.175'}).
{op: add, count: 1, id: 1000, time: 78ms}
ok
(node6@10.93.26.175)7> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 71ms, failed: 0, timeout: 0}
ok
(node6@10.93.26.175)8> test:add(1, Keys, {node, 'node1@10.93.26.175'}).
{op: add, count: 1, id: 1000, time: 99ms}
ok
(node6@10.93.26.175)9> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 90ms, failed: 0, timeout: 0}
ok
(node6@10.93.26.175)10>
```

```
ok
(node7@10.93.26.175)6> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 65ms, failed: 0, timeout: 0}
ok
(node7@10.93.26.175)7> test:add(1, Keys, {node, 'node1@10.93.26.175'}).
{op: add, count: 1, id: 1000, time: 98ms}
ok
(node7@10.93.26.175)8> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 91ms, failed: 0, timeout: 0}
ok
(node7@10.93.26.175)9>
```

```
(node8@10.93.26.175)6> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 65ms, failed: 0, timeout: 0}
ok
(node8@10.93.26.175)7> test:add(1, Keys, {node, 'node1@10.93.26.175'}).
{op: add, count: 1, id: 1000, time: 81ms}
ok
(node8@10.93.26.175)8> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 80ms, failed: 0, timeout: 0}
ok
(node8@10.93.26.175)9>
```

Figure 4: Two node ring test.



```
[14] 1 zsh 2 erl 3 nano
       (l)oaded (v)ersion (k)ill (D)b-tables (d)istribution
^C%

Homework/HW5 on main [!?] took 57s
→ erl -name node1@10.93.26.175 -setcookie chordy -connect_all false
Erlang/OTP 28 [erts-16.0.3] [source] [64-bit] [smp:10:10] [ds:10:10:10] [async-threads:1] [jit] [dtrace]

Eshell V16.0.3 (press Ctrl+G to abort, type help(). for help)
(node1@10.93.26.175)1> W1 = test:start(node2).
<0.93.0>
(node1@10.93.26.175)2> register(node, W1).
true
(node1@10.93.26.175)3> W1 ! probe.
probe
19407083: pre: {766205459,<10990.94.0>} suc: {766205459,<10990.94.0>}, storage: 0
probe: passed through ring [19407083,766205459] (2) in 2344μs
(node1@10.93.26.175)4> W1 ! probe.
probe
19407083: pre: {766205459,<10990.94.0>} suc: {766205459,<10990.94.0>}, storage: 1026
probe: passed through ring [19407083,766205459] (2) in 4722μs
(node1@10.93.26.175)5>
```

```
BREAK: (a)bort (A)bort with dump (c)ontinue (p)roc info (i)nfo
       (l)oaded (v)ersion (k)ill (D)b-tables (d)istribution
^C%

Homework/HW5 on main [!?] took 10m15s
→ erl -name node3@10.93.26.175 -setcookie chordy -connect_all false
Erlang/OTP 28 [erts-16.0.3] [source] [64-bit] [smp:10:10] [ds:10:10:10] [async-threads:1] [jit] [dtrace]

Eshell V16.0.3 (press Ctrl+G to abort, type help(). for help)
(node3@10.93.26.175)1> W2 = test:start(node2, {node, 'node1@10.93.26.175'}.
* 1:52: syntax error before: '.'
(node3@10.93.26.175)1> W2 = test:start(node2, {node, 'node1@10.93.26.175'}).
<0.94.0>
766205459: pre: {19407083,<10692.93.0>} suc: {19407083,{node,'node1@10.93.26.175'}}, storage: 0
766205459: pre: {19407083,<10692.93.0>} suc: {19407083,{node,'node1@10.93.26.175'}}, storage: 2974
(node3@10.93.26.175)2>
```

```
26801732: pre: {939860128,<10692.93.0>} suc: {426855416,<11028.93.0>}, storage: 0
26801732: pre: {939860128,<10692.93.0>} suc: {426855416,<11028.93.0>}, storage: 0
26801732: pre: {939860128,<10692.93.0>} suc: {426855416,<11028.93.0>}, storage: 0
26801732: pre: {939860128,<10692.93.0>} suc: {426855416,<11028.93.0>}, storage: 340
(node3@10.93.26.175)2>
BREAK: (a)bort (A)bort with dump (c)ontinue (p)roc info (i)nfo
       (l)oaded (v)ersion (k)ill (D)b-tables (d)istribution
^C%

Homework/HW5 on main [!?] took 10m13s
→
```

```
426855416: pre: {26801732,<11029.93.0>} suc: {852614199,<11028.94.0>}, storage: 0
426855416: pre: {26801732,<11029.93.0>} suc: {852614199,<11028.94.0>}, storage: 0
426855416: pre: {26801732,<11029.93.0>} suc: {852614199,<11028.94.0>}, storage: 1601
(node4@10.93.26.175)2>
BREAK: (a)bort (A)bort with dump (c)ontinue (p)roc info (i)nfo
       (l)oaded (v)ersion (k)ill (D)b-tables (d)istribution
^C%

Homework/HW5 on main [!?] took 10m11s
→
```

```
{op: add, count: 1, id: 1000, time: 109ms}
ok
(node5@10.93.26.175)5> test:add(1, Keys, {node, 'node1@10.93.26.175'}).
{op: add, count: 1, id: 1000, time: 118ms}
ok
(node5@10.93.26.175)6> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 109ms, failed: 0, timeout: 0}
ok
(node5@10.93.26.175)7> test:add(1, Keys, {node, 'node1@10.93.26.175'}).
{op: add, count: 1, id: 1000, time: 69ms}
ok
(node5@10.93.26.175)8> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 66ms, failed: 0, timeout: 0}
ok
(node5@10.93.26.175)9> test:add(1, Keys, {node, 'node1@10.93.26.175'}).
{op: add, count: 1, id: 1000, time: 112ms}
ok
(node5@10.93.26.175)10> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 90ms, failed: 0, timeout: 0}
ok
(node5@10.93.26.175)11>
```

```
(node6@10.93.26.175)5> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 90ms, failed: 0, timeout: 0}
ok
(node6@10.93.26.175)6> test:add(1, Keys, {node, 'node1@10.93.26.175'}).
{op: add, count: 1, id: 1000, time: 78ms}
ok
(node6@10.93.26.175)7> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 71ms, failed: 0, timeout: 0}
ok
(node6@10.93.26.175)8> test:add(1, Keys, {node, 'node1@10.93.26.175'}).
{op: add, count: 1, id: 1000, time: 99ms}
ok
(node6@10.93.26.175)9> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 90ms, failed: 0, timeout: 0}
ok
(node6@10.93.26.175)10>
```

```
ok
(node7@10.93.26.175)6> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 65ms, failed: 0, timeout: 0}
ok
(node7@10.93.26.175)7> test:add(1, Keys, {node, 'node1@10.93.26.175'}).
{op: add, count: 1, id: 1000, time: 98ms}
ok
(node7@10.93.26.175)8> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 91ms, failed: 0, timeout: 0}
ok
(node7@10.93.26.175)9>
```

```
(node8@10.93.26.175)6> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 65ms, failed: 0, timeout: 0}
ok
(node8@10.93.26.175)7> test:add(1, Keys, {node, 'node1@10.93.26.175'}).
{op: add, count: 1, id: 1000, time: 81ms}
ok
(node8@10.93.26.175)8> test:check(1, Keys, {node, 'node1@10.93.26.175'}).
{op: look, count: 1, id: 1000, time: 80ms, failed: 0, timeout: 0}
ok
(node8@10.93.26.175)9>
```

Figure 5: Four node ring test.