Dynamo Simulator

Karanbir Singh Chahal (ksc487), Udit Arora (ua388)

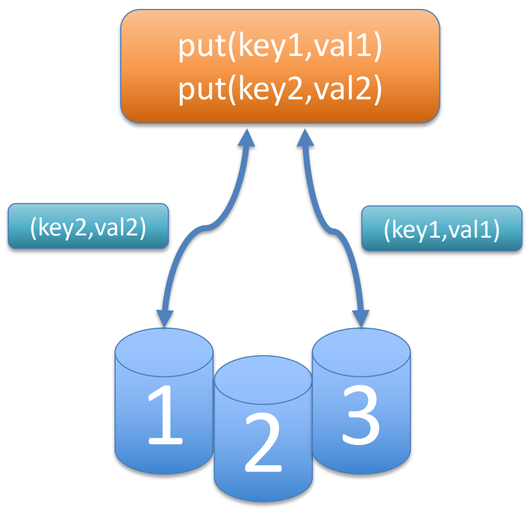
https://github.com/karanchahal/dynamo-clone

PROBLEM STATEMENT

Dyanamo: A weakly-consistent distributed key-value store.

Motivation: Reliably storing and accessing data in a distributed environment is important for modern web-scale applications.

Dyanamo showed how an eventually-consistent distributed key-value store can be built using different techniques while handling failures, which allows it to be used in production for demanding problems. We aim to **implement** and **analyze** this.



EXISTING / RELATED WORK

DeCandia at al from Amazon proposed Dynamo and implemented it for use internally by various services that required a high availability key-value store. They detailed the techniques used and analyzed the performance in different settings.

Our work implements Dynamo in a modern language along with a simulation framework to carry out different analysis. We also develop a web-based front-end for simulation.

APPROACH

We use Python process as Dynamo nodes with GRPC-based communication between them. We also build a web-app based simulation framework using Python-Flask for experimentation with different network and system configurations.

We implement the following protocols:

- Partitioning using **consistent hashing**
- **Vector clocks** for reconciliation of reads
- Handling temporary failures using **hinted handoff** and **sloppy quorum**
- Recovering from permanent failures via an **anti-entropy mechanism**
- **Gossip protocol** for failure detection

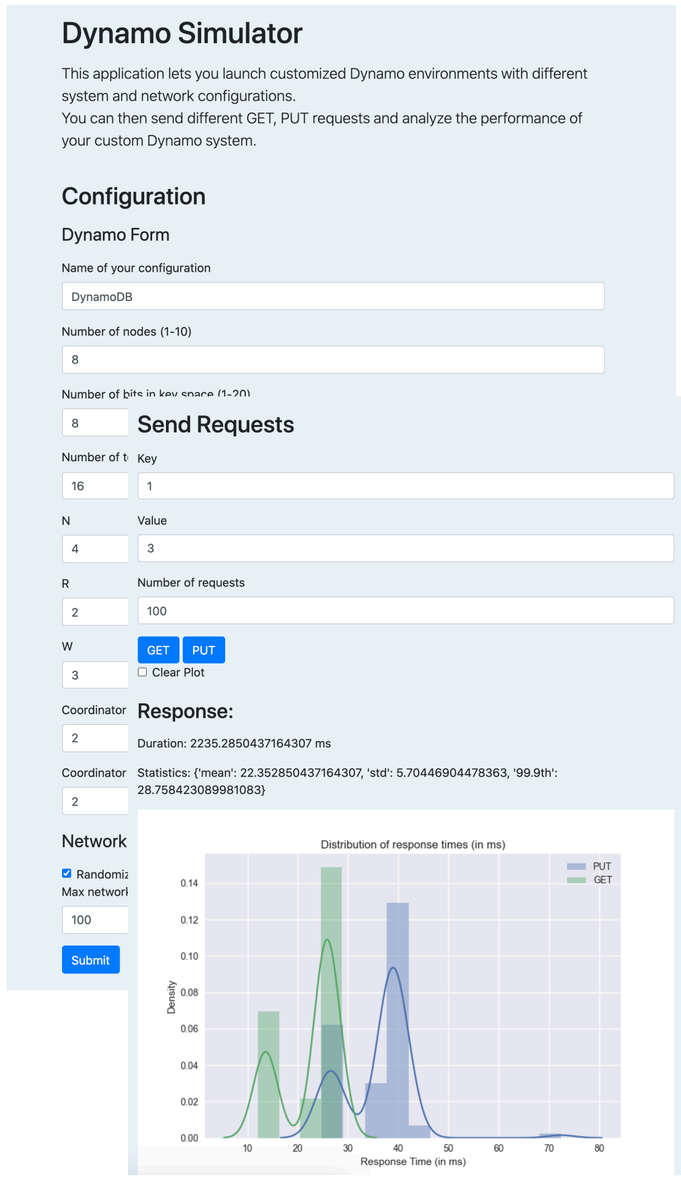
DESIGN AND IMPLEMENTATION

Our implementation primarily consists of:

- DynamoNode
- Launcher
- Client
- Web simulator

We built each component in a modular manner, with unit-tests across modules to check the correctness.

The diagrams on the right shows the simulator that lets users launch customized Dynamo instances and send different requests to it.



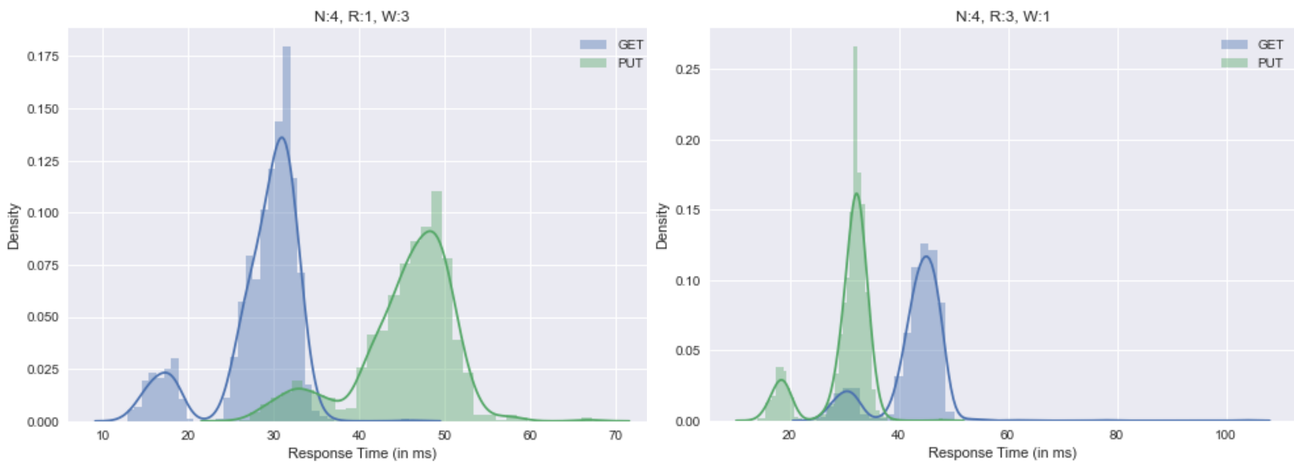
RESULTS

In the following experiment, we analyze 2 configurations which favor reads or writes. The results are obtained from 1000 requests sent to random keys in batches of 10, with network latency of 10 ms.

We observe that Dynamo can be configured for different use cases.

Configuration	Get Response (mean)	Get Response (std dev)	Get response (99.9th %ile)
N: 4, R: 1, W: 3	29.22 ms	7.00 ms	36.68 ms
N: 4, R: 3, W: 1	42.78 ms	5.88 ms	77.67 ms

Configuration	Put Response (mean)	Put Response (std dev)	Put response (99.9th %ile)
N: 4, R: 1, W: 3	44.84 ms	5.57 ms	66.28 ms
N: 4, R: 3, W: 1	30.13 ms	5.10 ms	38.91 ms



CONCLUSIONS, FUTURE WORK

We implemented Dynamo from scratch and presented a web app to simulate and analyze different loads under different configurations.

Future work

- To improve load-balancing, we can use multiple techniques like auto-scaling, routing of requests to other top-N nodes in the ring other than the first.
- Improve simulation front-end to support more types of analysis.