

BlinkDB Benchmarking Design

Document

1. Introduction

BlinkDB is a lightweight, high-performance key-value store that supports the Redis wire protocol (RESP-2) and handles multiple concurrent client connections efficiently using kqueue (for macOS/BSD) or epoll (for Linux). This document outlines the design decisions made for implementing BlinkDB's storage engine, connection management layer, and client-server communication using kqueue.

2. Architecture Overview

The BlinkDB system consists of:

- **Storage Engine:** In-memory key-value store with append-only file (AOF) persistence.
- **TCP Server:** Handles multiple client connections using asynchronous I/O.
- **Connection Management:** Uses kqueue to handle multiple events efficiently.
- **RESP-2 Protocol Parser:** Implements a parser to support Redis-like commands (SET, GET, DEL).

3. Storage Engine (BlinkDB)

BlinkDB's storage engine is designed for read-intensive workloads with efficient lookup operations.

Key Features

- Uses an **unordered_map** for $O(1)$ lookups.
- Implements **mutex locking** to handle concurrent access.

- **Append-Only File (AOF)** for durability.

Implementation Details

- **SET key value:** Stores a key-value pair in memory and appends it to the AOF file.
- **GET key:** Retrieves the value associated with a key.
- **DEL key:** Deletes a key from memory and appends the deletion to the AOF file.

Example Usage

```
SET user Krishna
```

```
GET user → "Krishna"
```

```
DEL user → OK
```

```
GET user → nil
```

4. Connection Management Layer

Design Choice: kqueue vs. Threads

Why kqueue?

- Scalable and event-driven.
- Efficient for handling multiple I/O events.
- Lower CPU overhead compared to multi-threading.

Alternatives Considered

- **Thread-per-connection model** (inefficient for large concurrent connections).
- **Epoll** (similar to kqueue, chosen for Linux systems).

Implementation Details

- Uses **non-blocking sockets** with kqueue for event-driven handling.
- Registers **EVFILT_READ** and **EVFILT_WRITE** filters for each client connection.
- Manages **client state** (read buffer, write buffer) efficiently.

Event Handling

1. **New Connection:** Accept and register client.
2. **Read Event:** Process command from the client buffer.
3. **Write Event:** Send response to the client.
4. **Close Event:** Clean up resources.

5. Example Execution

Here's an example of BlinkDB in action:

AOF Log (**blinkdb.aof**)

```
SET user Krishna
```

```
SET password helloworld
```

```
DEL user
```

Terminal Execution

```
127.0.0.1:9000> SET user Krishna
```

```
OK
```

```
127.0.0.1:9000> SET password helloworld
```

```
OK
```

```
127.0.0.1:9000> GET user
```

```
"Krishna"
```

```
127.0.0.1:9000> DEL user
```

```
(integer) 1
```

```
127.0.0.1:9000> GET user
```

```
(nil)
```

This confirms that:

- The **SET** command stores values correctly.
- The **GET** command retrieves values successfully.
- The **DEL** command removes values from the store.

6. Benchmarking with redis-benchmark

We evaluate BlinkDB using the redis-benchmark tool with varying concurrency and parallel connections.

Evaluation Setup

1. **Run BlinkDB on Port 9000**
2. **Execute Benchmark Tests**
 - `redis-benchmark -h 127.0.0.1 -p 9001 -t set,get -n 10000 -c 10`
 - `redis-benchmark -h 127.0.0.1 -p 9001 -t set,get -n 100000 -c 100`
 - `redis-benchmark -h 127.0.0.1 -p 9001 -t set,get -n 1000000 -c 1000`
3. **Measure Latency, Throughput, and Response Time.**

Expected Outcomes

- **Low latency** (~1 ms for **GET** queries).
- **High throughput** (~100,000 requests/sec for **GET**).
- **Stable performance** across different concurrency levels.

8. Benchmarking Methodology

Performance tests were conducted using **10, 100, and 1000 parallel clients**, executing **SET** and **GET** operations with a **3-byte payload**. The tests measured **throughput** and **latency** under various load conditions.

9. Benchmark Results

10 Parallel Clients

SET Operation

Requests	Throughput (req/sec)	Avg Latency (ms)
10,000	28,985.51	0.337
1,00,000	34,094.78	0.290
10,00,000	34,799.55	0.284

GET Operation

Requests	Throughput (req/sec)	Avg Latency (ms)
10,000	1,69,491.53	0.037
1,00,000	1,66,666.66	0.038
10,00,000	1,60,513.64	0.040

100 Parallel Clients

SET Operation

Requests	Throughput (req/sec)	Avg Latency (ms)
10,000	35,842.29	2.765
1,00,000	31,525.85	3.166
10,00,000	34,971.15	2.856

GET Operation

Requests	Throughput (req/sec)	Avg Latency (ms)
10,000	1,78,571.42	0.290
1,00,000	1,20,627.27	0.477
10,00,000	1,64,365.55	0.319

1000 Parallel Clients

SET Operation

Requests	Throughput (req/sec)	Avg Latency (ms)
10,000	29,673.59	31.295
1,00,000	35,298.27	28.097
10,00,000	34,660.84	28.826

GET Operation

Requests	Throughput (req/sec)	Avg Latency (ms)
10,000	1,38,888.89	4.294
1,00,000	1,69,491.53	3.004
10,00,000	1,71,791.78	2.940

10. Conclusion

BlinkDB demonstrates **high throughput** and **low latency** across different concurrency levels. The results indicate **efficient performance** under various load conditions, making it a **competitive choice for high-performance in-memory data storage**.