**Title:** Redis LATENCY GRAPH

**Excerpt:** The Redis latency monitoring framework offers a way to identify latency spikes related to real-time applications. It is possible to generate an ASCII-styled latency graph using the ‘LATENCY GRAPH’ command. This graph is useful to identify the latency trends and patterns in your system.

**Permalink:** redis-latency-graph

**Category:** Redis

# Redis Latency

Redis is a widely used in-memory data store that is well optimized for higher performance. Its speed makes it useful for real-time applications, caching, and session storage. At the same time, Redis offers several commands that are fast and performance-optimized. Still, some Redis commands take O(N) slower time complexity. The single-threaded nature of Redis raises latency concerns as well. Redis latency can be categorized into three main areas: client latency, command latency, and round-trip latency.

## Client latency

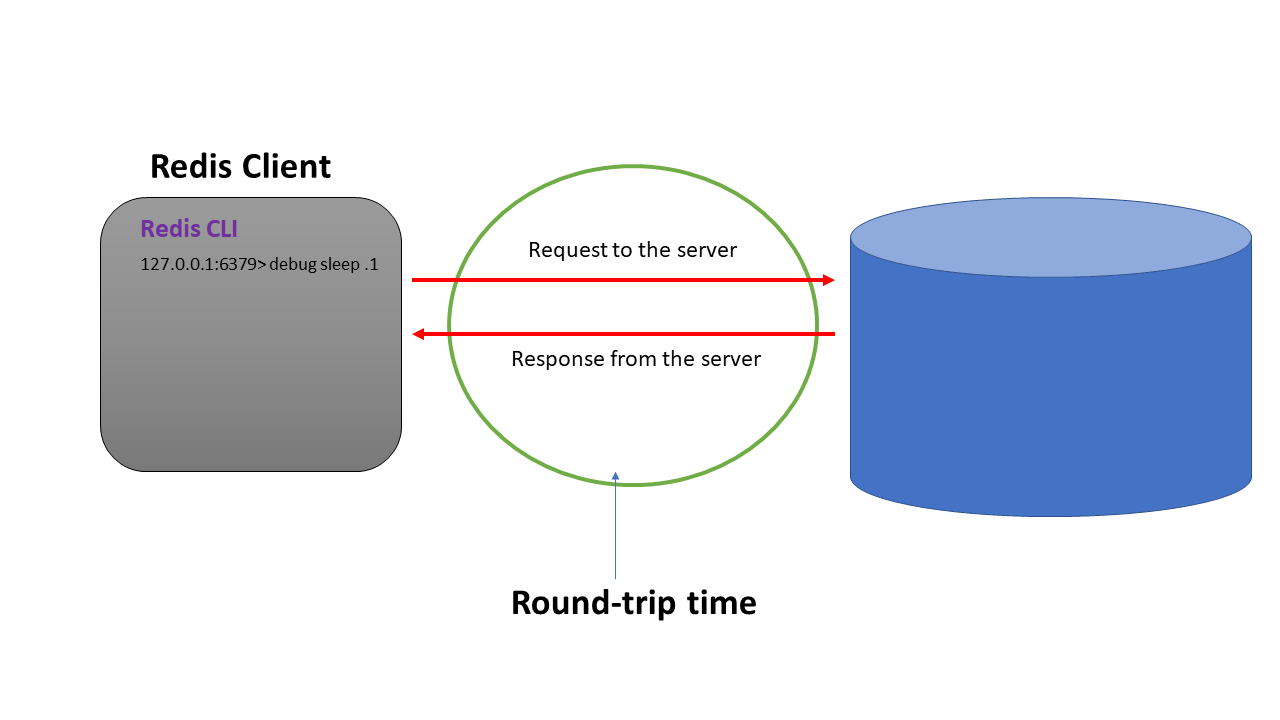
Redis comes with a client-server architecture. In some cases, multiple clients may try to connect to the Redis server at the same time. Since Redis is single-threaded, this will introduce a client queue where the server serves one client process at a given time. Hence, the concurrency latency arose. Therefore, later clients might need to wait.

## Command latency

Every command takes some time to execute. It can vary from microsecond to seconds. Hence, it has been identified as a latency source. Most of the Redis commands take constant or logarithmic time complexity. At the same time, some commands take O(N) time complexity. They are considerably slow.

## Round-trip latency

Round-trip is the time it takes to get a response from the Redis server after a command has been executed on the client-side. There might be different causes for round-trip latency such as network slowness, fork operations, and OS paging.



# Redis Latency Monitoring

Real-time applications use Redis where the performance is crucial. Hence, it is rewarding to have insights on Redis latency that will be helpful to take measures in advance. From version 2.8.13, Redis has added the latency monitoring component to its toolbox. This component is capable of recording latency spikes per event or specific code path.

## Latency events or code paths

The latency events(code paths) are nothing more than the generic or specific operations performed by Redis such as generic commands, fork system calls, unlink system, etc. When it comes to the generic commands, there are two main events defined by Redis.

1. command
2. fast-command

The ‘fast-command’ event is defined for Redis commands that have O(1) or O(log N) time complexity such as HSET, HINCRBY, HLEN, etc.

The ‘command’ code path measures the latency spikes for the other commands which have O(N) time complexity.

## Enabling latency monitoring in Redis server

The latency values depend on the application's nature. One application might consider the 10 milliseconds as high latency. At the same time, another application considers the 1 second as a high value. Hence, Redis offers you an option to define the latency threshold in the server. By default, the threshold value is 0. There are two ways to set this value in Redis.

1. Using the ‘CONFIG SET’ subcommand in runtime
2. Modifying the Redis configuration file

* The CONFIG SET subcommand

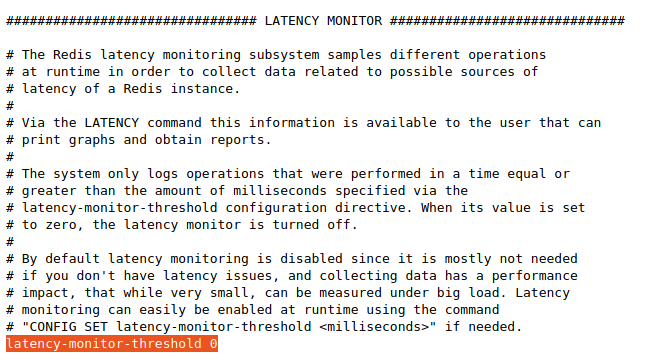
You can use the config set sub command with the parameter and its value to set the threshold value as shown in the following. Here, we set it as 500 milliseconds.

| config **set** latency-monitor-threshold 500 |
| --- |



* Modifying the redis.conf file

We can start the Redis server by providing all the configurations in a configuration file called 'redis.conf'. In the ‘LATENCY MONITOR’ section, you can set the ‘latency-monitor-threshold’ parameter value accordingly.



It is recommended to restart the Redis server after modifying the configuration file.

## The LATENCY GRAPH sub command

The ‘LATENCY’ command offers several sub commands to retrieve event-based latency information. One of the popular commands is ‘LATENCY GRAPH’. It draws a graph against the time when the event has happened. This graph is based on ASCII symbols and ranges from minimum latency value to maximum. The latency spikes are normalized between min and max latencies.

Let’s use the ‘debug sleep’ command to check how the latency graph information is generated.

### Syntax

| LATENCY GRAPH <**event\_name**> |
| --- |

The ‘event\_name’ parameter can be any event defined by the Redis latency monitoring framework such as command, fast-command, fork, etc.

### Example 01 - Applications with latency below the threshold

Let’s use the ‘debug sleep’ command to generate some latency spikes. It will go to sleep until the specified timeout. Since the latency threshold is 500ms, we will be issuing sleep commands with a timeout lower than the 500ms.

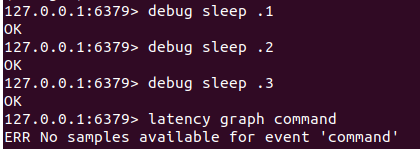
| **debug** sleep .1 **debug** sleep .2 **debug** sleep .3 |
| --- |

Next, we will issue the latency graph command as shown in the following.

| latency graph **command** |
| --- |

That would ideally generate the ASCII-style latency graph for the previous commands. Since the command execution time is lower than the threshold value in all the three ‘debug sleep’ commands, Redis will not generate latency spikes. If we assume this is our real-time application, you are all good. There are no latency issues attached.

Output:



As expected, it says no samples are available for this particular event.

### Example 02 - Applications with latency greater than the threshold

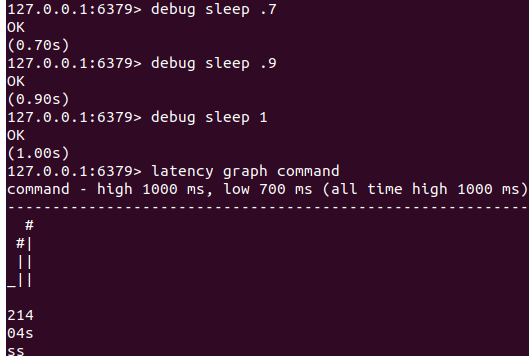
Let’s issue some debug commands with a timeout value that is greater than the threshold value. Usually, it is better to reset all the previous latency spikes before the next set of commands as shown in the following.

| latency reset **command** |
| --- |

Next, we will be issuing the debug sleep commands with a timeout value of more than 500ms.

| **debug** sleep .7 **debug** sleep .9 **debug** sleep 1 |
| --- |

Output:



As expected, the ASCII-styled graph has been generated by Redis. The ‘\_’ denotes the lowest latency value and the ‘#’ symbol denotes the highest latency spike that occurred within the last 20 seconds. This graph can be interpreted vertically. Each column is for an event that has occurred within the last seconds, minutes, or days. The left-most column can be interpreted as the event that happened 20 seconds ago, the next one has happened 14 seconds ago and the last column denotes an event that occurred 4 seconds ago.

# Conclusion

Redis is used as a data store for real-time applications. Therefore, the performance aspects are crucial. The latency monitoring framework is a component offered by Redis to monitor the latency spikes for predefined events. The 'latency graph command' can be used to generate the ASCII-styled latency spikes for a given time frame. It is used to identify the latency trends in your application and take necessary actions in advance. The latency spikes will be generated if and only if the latency time is greater than the threshold value. The latency threshold value might differ from one application to another based on the nature.