network

MDPI

*Article*

# A Performance Evaluation of In-Memory Databases Operations in Session Initiation Protocol

**Ali Al-Allawee [1,2], Pascal Lorenz [1,*], Abdelhafid Abouaissa [1] and Mosleh Abualhaj [3]**

[1] IRIMAS, University of Haute Alsace, 68000 Colmar, France
[2] Computer Science Department, University of Mosul, Mosul 41001, Iraq
[3] Department of Networks and Cybersecurity, Al-Ahliyya Amman University, Amman 19628, Jordan
* Correspondence: pascal.lorenz@uha.fr

**Abstract:** Real-time communication has witnessed a dramatic increase in recent years in user daily usage. In this domain, Session Initiation Protocol (SIP) is a well-known protocol found to provide trusted services (voice or video) to end users along with efficiency, scalability, and interoperability. Just like other Internet technology, SIP stores its related data in databases with a predefined data structure. In recent, SIP technologies have adopted the real advantages of in-memory databases as cache systems to ensure fast database operations during real-time communication. Meanwhile, in industry, there are several names of in-memory databases that have been implemented with different structures (e.g., query types, data structure, persistency, and key/value size). However, there are limited resources and poor recommendations on how to select a proper in-memory database in SIP communications. This paper provides recommended and efficient in-memory databases which are most fitted to SIP servers by evaluating three types of databases including Memcache, Redis, and Local (OpenSIPS built-in). The evaluation has been conducted based on the experimental performance of the impact of in-memory operations (store and fetch) against the SIP server by applying heavy load traffic through different scenarios. To sum up, evaluation results show that the Local database consumed less memory compared to Memcached and Redis for read and write operations. While persistency was considered, Memcache is the preferable database selection due to its 25.20 KB/s for throughput and 0.763 s of call–response time.

**Keywords:** SIP; in-memory; cache; NoSQL; memcache; Redis



**Citation:** Al-Allawee, A.; Lorenz, P.; Abouaissa, A.; Abualhaj, M. A Performance Evaluation of In-Memory Databases Operations in Session Initiation Protocol. *Network* **2023**, *3*, 1–14. https://doi.org/10.3390/network3010001

Academic Editors: Alessio Giorgetti and Alexey Vinel

Received: 11 November 2022
Revised: 5 December 2022
Accepted: 23 December 2022
Published: 28 December 2022



## 1. Introduction

Recently, the Internet has grown so rapidly with the big involvement of multimedia communication of end users' devices. That is, it became a daily requirement in ever-increasing applications such as social media, education, businesses, health, and other sectors. The IP Multimedia Subsystem (IMS) is an architecture that has been used to control and handle such kinds of multimedia communication [1]. Generally, the main aim of an IMS system is to offer network-controlled multimedia services such as voice and video communication. Specifically, IMS utilizes Session Initiation Protocol (SIP) [2] as a session control protocol that handles end-to-end call setup. SIP gains much consideration in communication technologies due to its simplicity, flexibility, and voice quality [3]. The main aim of SIP is to handle call sessions by initiating, manipulating, and terminating calls between end-to-end devices.

SIP technologies provide essential concepts of session control such as user registration, user location, call routing/forwarding, transaction/dialog management, and session negotiation [4]. Just like other internet technologies, SIP stores related data (user location, user-ID, passwords, flags, and domains) in a database server. Usually, by default, a database is co-hosted with an SIP server on the same machine, to ensure interoperability [5]. However, real-time services require SIP to operate with high performance to offer high service

quality. One of the performance degradations is that the database obtains a penalty during real-time communication [6]. For this, the NoSQL in-memory database is an alternative solution to improve data retrieval by caching the important and most frequently used data in memory [7,8].

In industry, there are over 25 NoSQL database systems deployed with different structures—to name a few, Memcached, Redis, Cassandra, Aerospike, Monogo, Hazelcast, Voldemort, RiakKV, etc. [9]. This research has selected three different cache database engines for evaluation, which are Local (OpenSIPS built-in) [10], Memcached [11], and Redis [12]. The selection is carried out based on a (1) Local cache engine deployed in a local shared memory in an SIP server (built-in). (2) Memcached is widely adopted by a range of Internet application companies (e.g., Facebook, Twitter, Zynga, and others) [13]. (3) Redis is also known to be utilized by Github, Snapchat, Weibo, and Flicker [14]. In particular, SIP (OpenSIPS) has already deployed the two databases as complete modules inside the server. Redis and Memcached are implemented inside the SIP server with high integrity and are well documented. By nature of the design, in-memory databases have different keys and value sizes, which yield various performance levels to SIP. In general, SIP technology uses this kind of database differently compared to web-based and cloud technologies. SIP only stores usernames/passwords during registration and retrieves (reads) them back for authentication. In other words, caching website components occurs in different ways of caching SIP data during real-time communication. The key problem in this research is that there are very limited guides or resources on how to select a recommended NoSQL database in SIP communication and how it affects SIP proxy performance.

The aim of this paper is to provide the recommended and most efficient in-memory database technique most suitable to SIP. The paper provides an experimental performance study on the impact of in-memory operations (store and fetch) against the SIP server. We evaluate the selected in-memory database engines through two different scenarios using three parameters: memory utilization, call response time, and throughput.

The paper begins by providing a background on SIP technology in Section 2. Details are provided about in-memory databases and their types in Section 3. Section 4 shows the implementation part followed by experimental results in Section 5. Finally, the conclusions are presented in Section 6.

## 2. Session Initiation Protocol

SIP is a signaling protocol defined and adopted by the SIP Working Group, under the umbrella of the Internet Engineering Task Force (IETF). The protocol was published with the first issue of RFC 3261 [15] toward the status of a proposed standard. SIP operates jointly with SDP (Session Description Protocol) [16], which is responsible for exchanging multimedia parameters such as IP version, port numbers, and voice codecs for RTP (Real-Time Transport Protocol) streams [17]. In general, the SIP server is commonly used to register new contact users, trigger and control multimedia sessions, modify ongoing calls, re-establish calls, and terminate calls [18].

One physical SIP server potentially consists of multi-logical servers on the same machine, each of which is reasonable for certain tasks. The main SIP elements involved in one SIP server device are below:

- Proxy Server:It is a network element that is responsible for routing incoming requests from a user agent and forwarding it to the next hop. SIP proxies route requests/responses to/from other entities based on URI.
- Registrar Server: It handles registration requests coming from user agents. When a new user tries to join the SIP network, the registrar server stores the user ID and password, URI, and location of the user in a database (database types explained in the next section). The registrar server aims to authenticate clients within their network.
- Redirect Server: It requests and looks up the destination possible addresses in the location database, which was originally created by the registrar server. The main aim

of the redirect server is to provide updated routing information to the proxy server. The redirect server stores such information in its associated database.

- Location Server: is another logical SIP server that is responsible for providing information about the possible destination to proxy and redirect servers. Destination user potentially updates their addresses regularly when the cases of the clients are in a mobility network or IP version translation (IPv6/IPv4). The location server stores all client's addresses in a database.

We can notice from above that all types of SIP servers are connected to local/remote SQL databases such as MySQL, MariaDB, Postgre, Berkeley, Oracle, Unix odbc, and many other names [19]. SIP core or modules can utilize the SQL services through the SQL interface. For instance, the Avpops module is an SQL interface that offers a direct query towards the SQL database.

In practice, SQL databases in all Internet applications, especially the real-time service, are considered to be an expensive operation, which made database hit disk penalty an existing issue in SIP technology; thus, NoSQL databases are found to make SIP life easier.

## 3. SIP with NoSQL

NoSQL (Not only SQL) is an umbrella to all database systems other than traditional SQL databases [20]. NoSQL databases store data models with a different structure than the traditional row and column table model out there in relational database management systems [21]. Unlike relational databases, key–value databases do not have a specified structure. Relational databases store their data in tables in which an individual column has a pre-assigned data type. However, NoSQL has four different categories of databases, which are key–value store, document-based, column-oriented, and graph databases. Each of these is designed for particular data types and requirements. According to SIP, key–value is the desired store structure to be used generally, enriched with a collection of keys and value pairs. On one hand, a key is a unique identifier for one or a set of values. On the other hand, values are real data that SIP retrieves from a database. Values can be any type of data—numbers, strings, or other complex data structures. For instance, URI has an identifier of the client as keys and their values can be example.sip.com, IPv4, and/or IPv6 [22].

Generally, the NoSQL database stores its data in different means of storage, disk or in-memory (RAM) storage. In-memory databases are purpose-built databases that utilize memory for data storage, unlike other databases which store data in disk (e.g., SSD) storage [23]. The main reason for SIP to use in-memory database storage is to operate with minimal response time during real-time communication—that is, to overcome the overhead of database operations (read/write) against the disk. Hitting a database during SIP communication is considered a high impact in terms of time. This is the primary fact that leads SIP servers to store the most necessary and frequent high-hit ratio data into cache [24].

The cache is a high-speed data storage layer which is the place where a subset of data is stored and is transient in nature. The aim of the cache is that, when the subsequence requests particular data, it will be provided faster than in the disk storage. Data reside in the cache storage for a period of time depending on the expiration time which can be assigned prior. Caching provides SIP servers with efficient data retrieval from the in-memory database. That, in turn, increases SIP performance with high-quality service for end users.

Technically , the SIP server deals with variant names of key–value cache modules, and in this research, three different structures will be used, and they are as follows:

- Local : it is a local cache database module used in the OpenSIPS server that resides in the server's shared memory. Because it is local, it performs very fast due to the low external communication. The local cache database requires no package installation, and it uses the same shared memory of OpenSIPS called shmem class.

However, the main issue beyond the Local cache is persistency; it loses all cached data once the server encounters any failure or restarts. OpenSIPS accesses the Local cache by using a key–value interface called cachedb_local. For example, to fetch a password from the local cache stored at the integer, attribute 55 can use: cache_fetch("local", "passwd_$tu", $avp(i:55))

- Memcached: is a well known in-memory key–value store system used relatively for light data storage and high-speed data retrieval. Memcached is considered one of the high-performance caching systems. The main advantage of the Memcached system is the ability to access data through multi-threaded connections. In other words, a single thread can be open for accepting a single connection to handle one client. Memcached copes with simple data structures, integers, and strings, Thus, to store other data types, pre-processed operations have to be performed to serialize to string or binary before storing. The good news of SIP is that it utilizes a simple data structure (integers and strings) in the database; thus, it requires no further data pre-processing in Memcached. The default size limit in Memcached is 250 characters for key, and 1 MB for value, the value size can be altered during compilation. For big data, Memcached provides LRU (Least Recently Used) methodology that aims to remove old data and free up for new data in cases when the memory is full. SIP can set the expiration time during the use of Memcached in the script.
  cache_store("memcached", "passwd_tu", "avp(i:55)", 1200); 1200 millisecond time to expire.
- Redis: found by Salvatore Sanfilippo [25], is a key–value store for cache, database, and message broker. Redis is a feature-rich data storage system that is able to support advanced data structures, snapshots, replication, transaction, and Pub/Sub, and well documented. It supports highly complex data types such as: strings, hashes, sets, and sorted sets. However, SIP technologies deal with low data complexity; thus, it does not enjoy all of the Redis features. Moreover, Redis supports a large key and value size, 512 MB. This guarantees high optimization in a hashing mechanism, leading to desired performance.
  Redis operates in a client/server mechanism, which is able to handle multi clients concurrently. It also supports sharing clustering mode, which is the case in which the client library hashes over distributed servers. In contrast to Memcached, Redis does not support multi-threaded connections even when clustering mode is used. Using the Pub/Sub feature in the SIP script can be achieved with the key–value interface: cache_sub("redis", "credit _fU", avp(cost), 0);

## 4. Related Works

Key–value cache systems have been studied and analyzed in several fields. Web servers have engaged in this evaluation as key–value cache systems are widely involved in such fields. Meanwhile, a number of research activities have been conducted for benchmarks and evaluation.

The work of [26] examined different NoSQL databases and evaluated their performance based on data storage and retrieval. In this work, a YCSB tool had been utilized to measure the performance of three names of databases which are MongoDB, Cassandra, and Redis. The authors found that MongoDB finally has a superior performance with the NoSQL database over Redis and Cassandra. MongoDB has significantly decreased the latency factor for all operation counts. The authors of [27] provided a comparative study between three solutions widely employed: Cassandra, Redis, and MongoDB by testing the run-time of read, update, scan, and read–modify–write operations. In summary, their results show that, with read and write operations, Redis outperforms other databases due to the use of volatile memory to store and retrieve data. On other hand, MongoDB performs with higher efficiency in the read operations than Cassandra due to the registers mapping technique of MongoDB, which helps increase read performance. In an empirical comparison study between Aerospike, Memcached, and Redis analyzed in [28],

the work aimed to present a comparison using variant workloads (read-heavy, balanced, and write-heavy). The research was conducted based on a multiple-client scenario and configured Memcached with two thread connections. The authors found that Memcached had the best performance over others due to the caching layer technique that has been used in read-heavy and balanced scenarios. In [29], the authors have examined five names of database systems in the field of health enterprises, and they have evaluated operation completion time and memory efficiency. The results showed that no winner database provides the best performance for all data operations. The authors in [30] have provided a benchmark on Yahoo! Cloud Serving with NoSQL database performance by applying 600,000 records in different workloads. The authors have evaluated five database names including Redis, HBase, Cassandra, OrientDB8, and MongoDB. They reported that Redis provides the best performance over others' column family databases, while HBase and Cassandra reported better update performance. The author of [31] had implemented a custom list data type in Memcached over Redis in the web industry, and the main finding of the thesis is that (1) Redis has superior performance with a small data size of the list and (2) when data size increases, Redis performance decreases. (3) Horizontal scalability in Memcached had better performance over Redis for a list of large data sizes. The work in [32] has examined four MySQL database machines according to access modes by the SIP server, and the authors analyzed the performance based on system recovery delay, registration response time, and processing delay. They also found that the response time is short in memory-only mode, and the response time of the write-back mode is the same as in memory-only mode. However, the write-through mode consumes extra time over memory-only mode.

At the time of writing, there are no studies or benchmarks completed in the field of SIP and NoSQL databases. In fact, the NoSQL database has gained wide consideration in the field of web industries and research, but not in SIP.

## 5. Evaluation Methodology

This section discusses the main methodology steps involved in the evaluation—by implementing/configuring the three in-memory key–value databases and workload, and then showing the testing scenario.

### *5.1. Implementation*

In-memory key–value databases have been implemented and configured in the SIP server. In this research, the OpenSIPS server is utilized as a test bed environment. OpenSIPS version 3.2 is an open-source server chosen to be a reliable, scalable, and fully compatible environment with database servers. In this testing, OpenSIPS functions as a proxy, location, and registrar server, all residing in a single physical machine (Debian 10 server). During the implementation phase, some noticeable challenges have been encountered. Thus, workaround configurations have been set in the OpenSIPS server to achieve the desired testing scenarios, the majority of which are:

The Memcached server has been installed on the same machine as the OpenSIPS server to eliminate network and external factors. Memcached version (1.5.6) has been installed along with libMemcached-tools to be used inside OpenSIPS script by pointing the SIP server to the Memcached server via port 11211 using: mod param("cachedb _Memcached", "cachedb _url", "Memcached: group1://127.0.0.1:11211/"). Redis version (5.0.3) is also installed with libhiRedis-dev on the same machine as SIP. The OpenSIPS server points to the Redis server through its default port 6379 using modparam("cachedb _redis", "cachedb_url", "redis: cluster1:// 127.0.0.1:6379/"). We disabled the cluster mode and snapshot option in the Redis server as it is out of this research scope. In addition, we have disabled the persistence option in Memcached and Redis servers.

### *5.2. Network Elements and Workload*

The testing environment has been conducted in a local network, and all testing elements are isolated in 1000 Mbps to avoid exterior impacts. In general, the OpenSIPS server includes three entities of SIP (proxy, redirect, and location) and three database servers (Memcached, Redis, and Local) in a single machine. An SIP server is connected to two legs—user agent client (UAC) and user agent server (UAS)— each of which is in an individual machine. The specification of all devices involved in this test is populated in Table 1:

**Table 1.** Network elements' specifications.

| Entity | SIP Proxy Server | UAC Machine | UAS Machine |
|---|---|---|---|
| Machine Model | HP EliteDesk | Dell-Vostro Laptop | Lenovo B570e Laptop |
| Operating System | Debian 10 Buster server | Debian Stretch version 9 | Debian Stretch version 9 |
| RAM | 16 Giga Byte | 2 Giga Byte | 4 Giga Byte |
| CPU | Intel Core i7-4790—3.6 GHz | 900@2.20 GHz | i3-2310M @ 2.10 GHz |
| SIP software | OpenSIPS 3.2 | SIPp 3.3 | SIPp 3.3 |
| Ethernet | 100 Mb/s | 100 Mb/s | 100 Mb/s |

The evaluation is conducted by applying heavy load traffic to gain stress SIP testing. In this research, SIPp [33] is utilized as an SIP traffic generator, which is a de facto standard testing tool that aims to generate SIP traffic toward SIP servers. SIPp is installed in end devices (UAC and UAS) individually. UAC (generator) sends traffic toward the SIP server, then to be received by the SIP server, and forwarded ahead to UAS (sink). SIPp at UAC is able to formulate the way of generating messages (Register or Invite) by modifying messages with highly customized XML language. In this paper, SIPp sends registration messages to perform user-name and password storage in the desired database. With other scenarios, SIPp sends an INVITE message to perform read from the database. The next subsection presents the possible scenarios of this research.

### *5.3. Testing Scenarios*

This research aims to evaluate the SIP server through different key–value databases. The evaluation was conducted in a variety ways to come up with a big number of possible scenarios.

#### 5.3.1. Topology

The SIP network builds in two different topologies: First, traffic sends from one UAC toward the SIP server; then, the SIP server forwards the traffic to one UAS. Second, traffic is generated from two separated UACs toward the SIP server and then forwards all traffic to one UAS. The aim of the second topology is to determine the impacts resulting from the multi-client connections. The test only uses two clients to isolate the database impact from the network connections. In addition, we have evaluated the SIP server only and not the clients. Thus, there is no variant notice in traffic distribution because, in the end, the SIP server will receive 1000 calls. Note that the term multi-clients does not mean cluster mode. Figure 1 shows testing topologies.

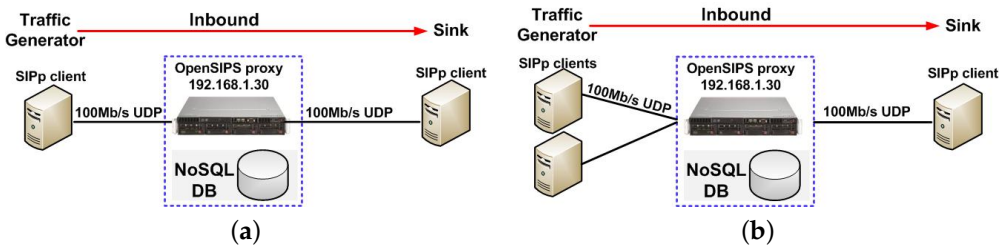


**Figure 1.** Testing topology (**a**) single client (**b**) multi clients.

#### 5.3.2. Load Scenario

SIPp generates SIP message traffic toward the SIP server starting with 100 calls per second (cps); later, the workload increased dramatically until 1000 cps, which is the desired load for database connections. Initially, SIPp runs for 1000 calls to warm up the testing environment prior to the actual test [34].

#### 5.3.3. Operations

Database operations have high impact on our evaluation; in this research, the Fetch operation is examined as database read, whereas Store operation is for data writing (store) in a database. Usually, SIP uses the Fetch operation to authenticate against user calls using cache_fetch. Store operation is used to register users for the first time by storing username and password via cache_store [35].

#### 5.3.4. Measurement Parameters

Evaluations were conducted based on three parameters including throughput (measures transmitted data per second), which is to measure data flow in kbps to provide results from a network perspective calculated using Equation (1):

$$\text{Thu} = \Sigma d * s/t + \Sigma d * r/t \quad (1)$$

where ds and dr are data send and receive, respectively, in the t second interval.

For memory usage, (RAM) examines the storage footprint of each database [36,37], OpenSIPS utilized shared memory (shmem) for better statistics, which can be calculated using Equation (2):

$$\text{Mem}(\%) = 100 - (((mf + b + c) * 100)/mv) \quad (2)$$

where mf = memory free, b = Buffer, c = memory cache, and mv = max value

Call response time (CRT) is calculated by measuring the round trip time in ms spend for a particular SIP message, and CRT is calculated using (3):

$$\text{CRT (ms)} = 200ok * Rt - Invite * St \quad (3)$$

where Rt and St are sending and receiving times, respectively.

## 6. Results and Discussion

This section discusses the main finding of this research by evaluating SIP servers against three different NoSQL key–value databases, namely, Local, Memcached, and Redis. The evaluation has been conducted over two different scenarios: one client and two clients. The parameters used in this test are Memory, Throughput, and Call response time.

### *6.1. Single Client*

Figure 2 shows the impact of key–value databases on OpenSIPS servers when it performs read (fetch) operation during call testing. Obviously, we can notice that the Local database consumes less memory than the Memcached and Redis, which is due to the high integrity between the OpenSIPS server and its own local database. However, the Local cache has a main structured drawback, which is persistency. The Local-database will lose the entire amount of data whenever the OpenSIPS server fails or restarts. Memcached comes second in this evaluation as it hits 6.61% of the memory due to OpenSIPS dealing with a very simple data structure suitable for Memcached. For overall load, Redis utilized the highest memory footprint. With the "store" operation, OpenSIPS aims to store some values in the in-memory database for current calls in real time, for instance, storing user passwords during registration. In all database structures, writing in the disk/memory requires extra effort compared to read operation. The reader can notice from Figure 3 that the three databases consume extra memory. Again, the Local database has higher

performance than others, with the same reason of read operation; Local database is a local special database for OpenSIPS. The memcached bar shows that the 0.56% of extra memory goes to writing operation and 0.58% in Redis.

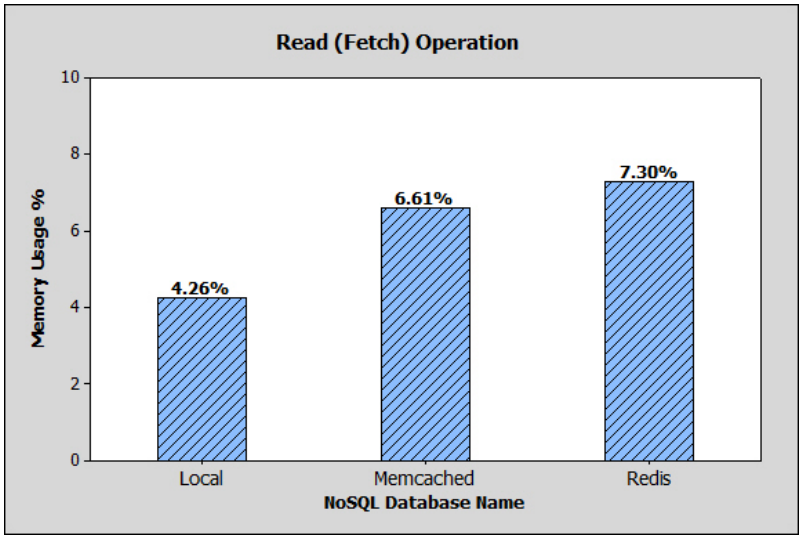


**Figure 2.** Memory usage with read operation (Single Client).

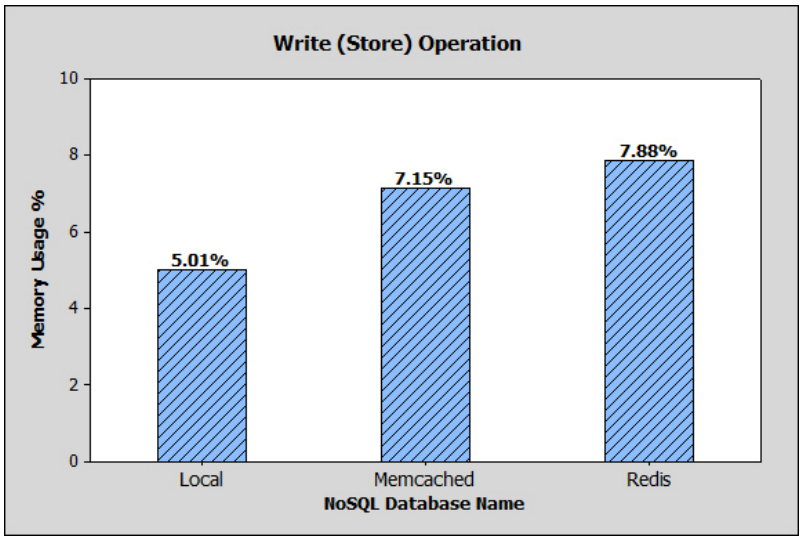


**Figure 3.** Memory usage with write operation (Single Client).

The throughput in the OpenSIPS server is measured based on the amount of network data being received for a period of 1 s. Clearly notice from Figure 4 that the Local database gains a high amount of data while processing fetch data, and 35.91 KB/s is the highest amount of data received in this process compared to other databases. This is due to the high integration and interoperation between the Local database in shared memory (shmem) and the OpenSIPS server. Memcached causes the OpenSIPS server to gain only 32.20 KB/s while reading data. This means that the reading process consumes extra reading time operation and made the OpenSIPS server slow in receiving data. Despite the high features and data structure support, Redis performs with lower throughput as the memory structure requires a higher footprint for data. The write operation in Figure 5 shows the decrease of data received from UAC as the location server requires extra effort to store UAC user name and password. Memcached and Redis both received around 25 KB/s, which is not too much different from read operation. In fact, the write operation costs the server a delay by receiving 7 KB/s in the Local database; in reality, this is still an acceptable difference

in real time as this only happened during the user registration session and not calling. In real-time communication, user experience is only noticed during the łexchanging of voices end to end. In other words, during the registration session, a user does not care much about the registration delay as it only occurred once. It is worth mentioning in this stage that the data received (throughput) are the SIP signaling messages such as INVITE, ACK, and BYE. Media (RTP) data are not processed and received in this research as it is out of the scope. Figure 6 presents the period of time spent on a round trip call, in particular, the time taken to complete a successful call session starting from the UAC, SIP server, and UAS, then backward to the source again. When it comes to a millisecond, there is a silly difference between the three databases; between Local and Memcached, it is less than one millisecond. This, in fact, is attributed to the power of the OpenSIPS engine, a scalable and fast server. Redis performs in a very close response time compared to Local and Memcached. Interestingly, results can also be seen from Figure 7, when store operation only delays the call with 0.012 ms for a Local database, 0.028 ms for Memcached, and 0.048 ms in Redis, which is the highest. These differences are still in the acceptable range because, as mentioned earlier, store operation is occurring rarely with users, registration, or password-change.

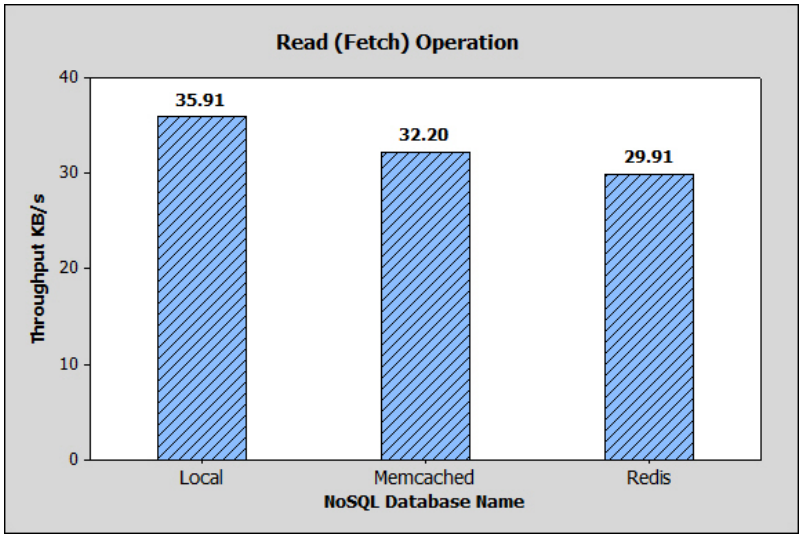


**Figure 4.** Throughput with read operation (Single Client).

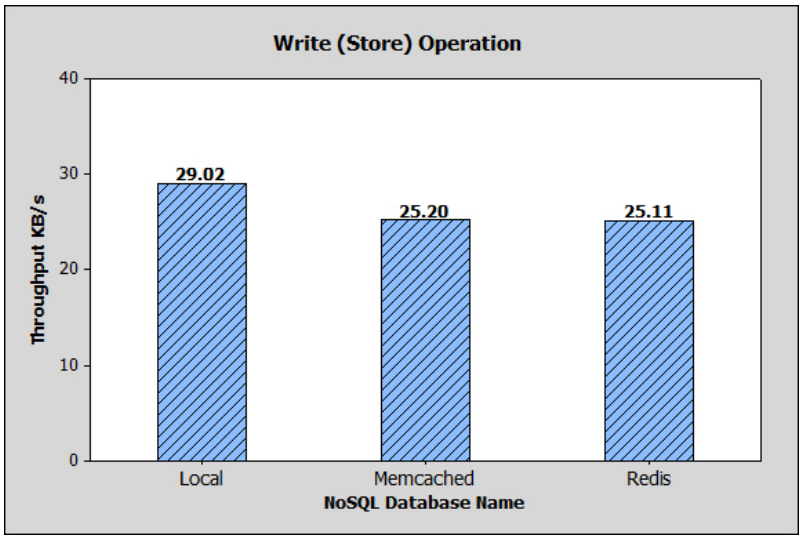


**Figure 5.** Throughput with write operation (Single Client).

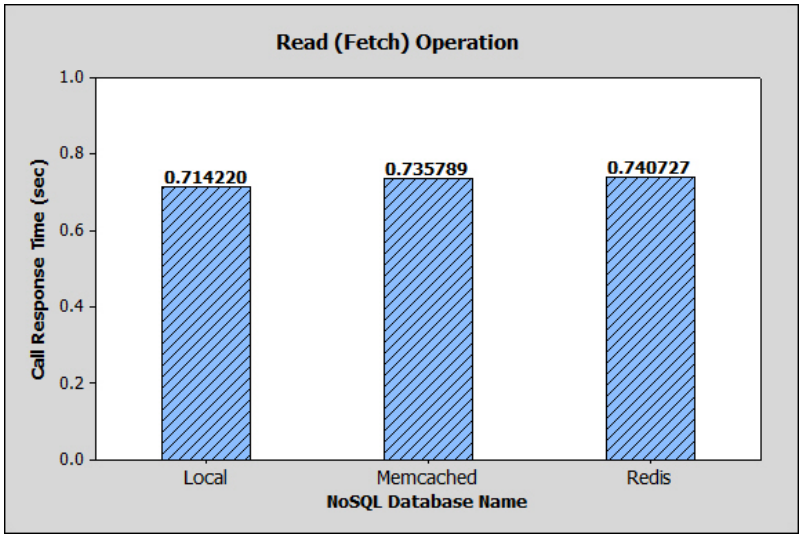


**Figure 6.** Call Response Time with read operation (Single Client).

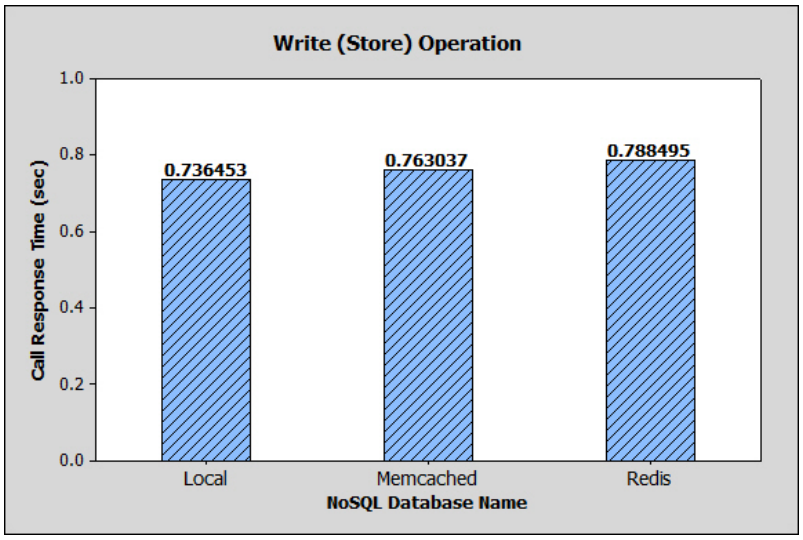


**Figure 7.** Call Response Time with write operation (Single Client).

### *6.2. Multi Clients*

This section presents the impact of multi clients accessing a single database (in OpenSIPS server) simultaneously. Two UACs generate the same load divided by two (500 calls for each) toward the SIP server. In this scenario, two clients read/write data from/in the NoSQL database at the same time. In Figure 8, the Local database uses 5.76% of memory. In contrast to the Single-client scenario, multi-clients consume an extra 1.5% of memory in read-operation. This can be attributed to the network setup and transport operations. Memcached performed as well as Local, which is considered acceptable performance compared to Redis. This is due to the multi-thread feature in Memcached, which allows for opening multi connections against clients. This feature is not available in the Redis database, despite it having a multi-feature. On another side, writing in multi-clients is not affected much, as the OpenSIPS server itself joins all the connections from upstream and points them to a single database. However, the extra memory usage is also attributed to network connection operation while serving the same amount of calls (500 for each).

In this scenario, throughput and Call Response time tests are omitted due to the redundant finding concept being there. Both tests have been conducted but obtained the same results of one client plus the network connection setup, which has already been found and discussed in Figure 9.

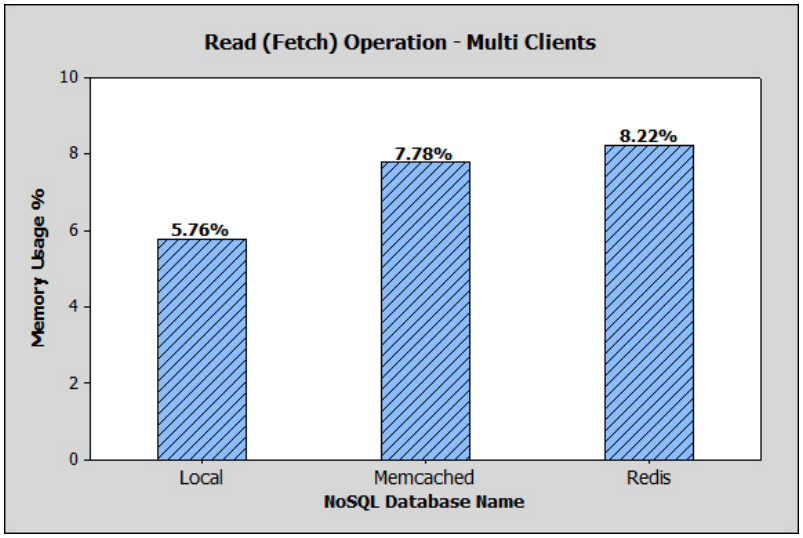


**Figure 8.** Memory usage with read operation (multi clients).

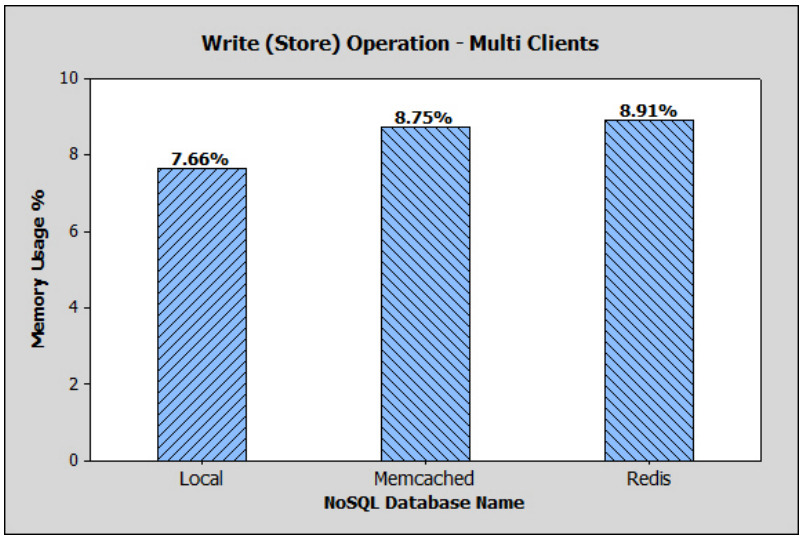


**Figure 9.** Memory usage with write operation (multi clients).

## 7. Conclusions

SIP service providers, in order to manage their users' provisions (username, password, domain name, etc), have to deal with variant means of database machines. SIP can deal with MySQL, NoSQL, and/or in-memory databases, all depending on the server's implementation. However, typical MySQL databases are too costly, especially for real-time communication, like SIP. Therefore, variant alternative databases have been found to improve database connection by SIP server performance such as the in-memory database, which utilizes the key–value model. Most databases known in the industry and research community are Memcached, Redis, and Local (inside the SIP server). However, there are limited resources and studies regarding in-memory databases with SIP performance, and how each database affects the SIP server. This paper discusses the main impact of a key–value database on an SIP server during session setup in real-time communication. The paper evaluated the performance of SIP servers in different scenarios: single-client and multi-clients. The evaluation tests were conducted through read and write database operations under different parameters. The test examined memory usage, network throughput, and call response time. The general finding of this research is that the Local database consumes less memory than Memcached and Redis. Memcached consumes 6.61% of the memory. In addition, we found that Memcached costs the server only 0.56% memory of

writing compared to the read operation. For throughput, the Local database, while processing fetch data, received 35.91 KB/s on the server side, which is the highest compared to other databases. Regarding response time, in store operation, the Local database spends 0.012 ms, 0.028 ms for Memcached, and 0.048 ms in Redis, which are all in the acceptable range. In summary, without persistency, a Local database is the best choice for SIP. Where persistency is considered, Memcached is a great choice here. Finally, if advanced data structure and rich features are required, Redis is a suitable selection for the future. Future work concerns wider database operations in cluster mode topology. In addition, a remote database server will be taken as a sharing in-memory database center for several nodes connecting.

**Author Contributions:** A.A.-A. and P.L.; methodology, A.A.-A., P.L., A.A.-A. and M.A.; validation, A.A.-A., P.L. and M.A.; formal analysis, A.A.-A., P.L., A.A. and M.A.; investigation, A.A.-A. and P.L.; data curation, A.A.-A. and P.L.; writing—original draft preparation, A.A.-A. and P.L.; writing—review and editing, A.A.-A., P.L., A.A. and M.A.; supervision, P.L. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** Not applicable.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

| | |
|---|---|
| CPS | Call per Second |
| CRT | Call Response Time |
| IETF | Internet Engineering Task Force |
| IMS | IP Multimedia Subsystem |
| IPv4 | Internet Protocol version 4 |
| IPv6 | Internet Protocol version 6 |
| LRU | Least Recently Used |
| MySQL | My Structured Query Language |
| NoSQL | Not Only Structured Query Language |
| RAM | Random Access Memory |
| RTP | Real-Time Transport Protocol |
| SDP | Session Description Protocol |
| SDP | Session Description Protocol |
| SHMEM | Shared Memory |
| SIP | Session Initiation Protocol |
| SIPp | SIP Performance |
| UAC | User Agent Client |
| UAS | User Agent Server |

## References

1. Ilyas, M.; Ahson, S.A. *IP Multimedia Subsystem (IMS) Handbook*; CRC Press: Boca Raton, FL, USA, 2018.
2. Barz, H.W.; Bassett, G.A. Session Initiation Protocol. In *Multimedia Networks: Protocols, Design and Applications*; Wiley Telecom: Hoboken, NJ, USA, 2016; pp. 147–182. [CrossRef]
3. Ahson, S.A.; Ilyas, M. *SIP Handbook: Services, Technologies, and Security of Session Initiation Protocol*; CRC Press: Boca Raton, FL, USA, 2018.
4. Semerci, M.; Cemgil, A.T.; Sankur, B. An intelligent cyber security system against DDoS attacks in SIP networks. *Comput. Netw.* **2018**, *136*, 137–154. [CrossRef]
5. Ali Abdulrazzaq, K.; Ali, A.K.; Praptodiyono, S. The Impact of Elliptic Curves Name Selection to Session Initiation Protocol Server. In Proceedings of the International Conference on Advances in Cyber Security, Penang, Malaysia, 8–9 December 2020; pp. 225–234.
6. Deng, C.; Li, G.; Zhou, Q.; Li, J. Guarantee the Quality-of-Service of Control Transactions in Real-Time Database Systems. *IEEE Access* **2020**, *8*, 110511–110522. [CrossRef]

7. Zhao, W.; Du, Y.; Zhang, M.; Liu, M.; Jin, K.; Ausavarungnirun, R. Application-Oriented Data Migration to Accelerate In-Memory Database on Hybrid Memory. *Micromachines* **2022**, *13*, 52. [CrossRef] [PubMed]
8. Zhang, K.; Ou, D.; Jiang, C.; Qiu, Y.; Yan, L. Power and Performance Evaluation of Memory-Intensive Applications. *Energies* **2021**, *14*, 89. [CrossRef]
9. Fang, J.; Mulder, Y.T.; Hidders, J.; Lee, J.; Hofstee, H.P. In-memory database acceleration on FPGAs: A survey. *VLDB J.* **2020**, *29*, 33–59. [CrossRef]
10. Bogdan, A. OpenSIPS the New Breed of Communication Engine. Available online: https://www.opensips.org/ (accessed on 19 August 2022).
11. Cheng, W.; Ren, F.; Jiang, W.; Zhang, T. Optimizing the Response Time of Memcached Systems via Model and Quantitative Analysis. *IEEE Trans. Comput.* **2021**, *70*, 1458–1471. [CrossRef]
12. Liu, Q.; Yuan, H. A High Performance Memory Key-Value Database Based on Redis. *J. Comput.* **2019**, *14*, 170–183. [CrossRef]
13. Ma, W.; Zhu, Y.; Li, C.; Guo, M.; Bao, Y. BiloKey : A Scalable Bi-Index Locality-Aware In-Memory Key-Value Store. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *30*, 1528–1540. [CrossRef]
14. Stjepanovic, D.; Savic, M.; Jokić, J.; Marić, S. Performance measurements of some aspects of multi-threaded access to key–value stores. In Proceedings of the 2015 23rd Telecommunications Forum Telfor (TELFOR), Belgrade, Serbia, 24–26 November 2015; pp. 831–834. [CrossRef]
15. Rosenberg, J.; Schulzrinne, H.; Camarillo, G.; Johnston, A.; Peterson, J.; Sparks, R.; Handley, M.; Schooler, E. *SIP: Session Initiation Protocol*; Technical Report; The Internet Society: Reston, VA, USA, 2002.
16. Begen, A.; Kyzivat, P.; Perkins, C.; Handley, M. SDP: Session Description Protocol, RFC 8866. 2021. Available online: https://www.rfc-editor.org/info/rfc8866 (accessed on 19 August 2022) [CrossRef]
17. Khudher, A.A. Sip aspects of ipv6 transitions: Current issuesand future directions. *J. Eng. Sci. Technol.* **2019**, *14*, 448–463.
18. Khudher, A.; Ramadass, S. I-TNT: Phone number expansion and translation system for managing interconnectivity addressing in SIP peering. *J. Eng. Sci. Technol.* **2015**, *10*, 174–183.
19. Livingston, K.A.; Chung, M.; Sawicki, C.M.; Lyle, B.J.; Wang, D.D.; Roberts, S.B.; McKeown, N.M. Development of a publicly available, comprehensive database of fiber and health outcomes: Rationale and methods. *PLoS ONE* **2016**, *11*, e0156961. [CrossRef]
20. Davoudian, A.; Chen, L.; Liu, M. A survey on NoSQL stores. *ACM Comput. Surv.* **2018**, *51*, 1–43. [CrossRef]
21. Sicari, S.; Rizzardi, A.; Coen-Porisini, A. Security & privacy issues and challenges in NoSQL databases. *Comput. Netw.* **2022**, *206*, 108828.
22. Khudher, A.A.; Munther, A.; Praptodiyono, S. Efficient IPv4-IPv6 translation mechanism for IMS using SIP proxy. *Int. J. Internet Protoc. Technol.* **2022**, *15*, 41–52. [CrossRef]
23. Khan, K.; Pasricha, S.; Kim, R.G. A survey of resource management for processing-in-memory and near-memory processing architectures. *J. Low Power Electron. Appl.* **2020**, *10*, 30. [CrossRef]
24. Laghari, A.A.; He, H.; Laghari, R.A.; Khan, A.; Yadav, R. Cache performance optimization of QoC framework. *EAI Endorsed Trans. Scalable Inf. Syst.* **2019**, *6*, e7. [CrossRef]
25. Singh, R.K.; Verma, H.K. Redis-Based Messaging Queue and Cache-Enabled Parallel Processing Social Media Analytics Framework. *Comput. J.* **2022**, *65*, 843–857. [CrossRef]
26. Abu Kausar, M.; Nasar, M.; Soosaimanickam, A. A Study of Performance and Comparison of NoSQL Databases: MongoDB, Cassandra, and Redis Using YCSB. *Indian J. Sci. Technol.* **2022**, *15*, 1532–1540. [CrossRef]
27. Seghier, N.B.; Kazar, O. Performance Benchmarking and Comparison of NoSQL Databases: Redis vs MongoDB vs Cassandra Using YCSB Tool. In Proceedings of the 2021 International Conference on Recent Advances in Mathematics and Informatics (ICRAMI), Tebessa, Algeria, 21–22 September 2021; pp. 1–6. [CrossRef]
28. Anthony, A.; Rao, Y.N.M. Memcached, Redis, and Aerospike Key-Value Stores Empirical Comparison. Available online: https://anthonyaje.github.io/file/An_empirical_evaluation_of_Memcached_Redis_and_Aerospike_kvstore_Anthony_Eswar.pdf (accessed on 22 August 2022).
29. Kabakus, A.T.; Kara, R. A performance evaluation of in-memory databases. *J. King Saud-Univ.-Comput. Inf. Sci.* **2017**, *29*, 520–525. [CrossRef]
30. Abramova, V.; Bernardino, J.; Furtado, P. Experimental evaluation of NoSQL databases. *Int. J. Database Manag. Syst.* **2014**, *6*, 1. [CrossRef]
31. Rajbhandari, P. Benchmarking a Custom List Data Type in Memcached against Redis. Ph.D. Thesis, University of Cincinnati, Cincinnati, OH, USA, 2016.
32. Chen, W.E.; Cheng, S.Y.; Ciou, Y.L. A Study on Effects of Different Access Modes on Database Performance for SIP Server. In Proceedings of the 2014 Tenth International Conference on Intelligent Information Hiding and Multimedia Signal Processing, Kitakyushu, Japan, 27–29 August 2014; pp. 902–906.
33. Richard, G. SIPp a Free Open Source Test Tool Traffic Generator. 2014. Available online: https://sipp.sourceforge.net/ (accessed on 22 August 2022).
34. Sanka, A.I.; Cheung, R.C. Efficient high performance FPGA based NoSQL caching system for blockchain scalability and throughput improvement. In Proceedings of the 2018 26th International Conference on Systems Engineering (ICSEng), Sydney, Australia, 18–20 December 2018; pp. 1–8.

35. Passing, L.; Then, M.; Hubig, N.C.; Lang, H.; Schreier, M.; Günnemann, S.; Kemper, A.; Neumann, T. SQL-and Operator-centric Data Analytics in Relational Main-Memory Databases. In Proceedings of the EDBT, Venice, Italy, 21–24 March 2017; pp. 84–95.
36. Munther, A.; Abdulrazzaq, A.; Abualhaj, M.M.; Almukhaini, G. Reduce memory consumption for internet traffic classification. *Int. J. Netw. Virtual Organ.* **2021**, *24*, 144–160. [CrossRef]
37. Abualhaj, M.M.; Al-Khatib, S.N. A New Method to Boost VoIP Performance Over IPv6 Networks. *Transp. Telecommun.* **2022**, *23*, 62–72. [CrossRef]