International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2019): 7.583

Leveraging the Core Technology Capabilities when Designing Large Scale, Data Intense Systems

Santosh S Deshmukh

PMP, SCM, MSBA, Senior Member, IEEE

Abstract: ***A "key-value pair" is a fundamental data structure that associates a unique identifier or key with a corresponding value. This pairing allows developers to organize and retrieve data efficiently, making it a widely used concept in various programming languages and data storage systems. This paper provides insight into the innovative way of storing and populating data for standalone or browser-based software applications using the traditional "key-value" pair but altered approach. This innovation is real-life implementation when XML or JSON was less popular and limited in software application implementation. This paper also explains the insights into "key-value" pair efficiency in storing and retrieving the data for large data-intensive applications like clinical, medicine, or health applications. The necessity of the application demands customized data structure design to meet the specific needs of the application behavior. To meet the specific need, an innovative "request-response" data structure was designed for the clinical case management application, however, this can be easily implemented for any software application if given protocols been followed. This is an extremely generic, adaptive, scalable, and efficient data structure design that can be used for data storage and retrieval of any application. This can be accommodated fully or partially in any application design.***

Keywords: Key Value Attributes, Relational Database, Data storing techniques, computing latency.

1. Introduction

Data is the necessity of every application. As the speed of information growth exceeds Moore's Law at the beginning of this new century, excessive data is making great troubles to human beings.[2] Storing data on persistent media and retrieving data efficiently is a key challenge for every software application. This challenge becomes more complex when the information is in tiny chunks and needs frequent trips to store into database and retrieve in efficient manner when requested by application. Applications such as healthcare, medicine, social, clinical, sales applications capture small amounts of data but in large transaction volume. Unlike the latest technologies like big data or map reducer is widely used in heterogeneous, variety and complex form of datasets, arriving in increasing volumes and with more velocity [1], were not coexist in old days. However, XML and JSON have their own flavor, but they are based on simple principle of "Key-Value" pair which refers as attribute and value. JSON is known and widely used for server-to-server communication and XML is widely used for client to server communication, the actual implementation may change. However, the core of the request and response is who the data belongs (key) and what is that data needs to communicate (value). The key serves as a reference, allowing the retrieval or modification of the associated value. The "Request-response" design in this paper is fundamentally based on "Key -value" pair but with innovative approach to achieve specific purpose.

2. Background

Consider a clinical, case or disease management application. A patient admitted in hospital, LTC (Long Term Care) center or SNF (Skilled Nursing Facility) etc. for chronic care treatment. A patient has a long disease, treatment, and medication history. A large number of care plans might be active. Patients have growing or intermittent chief complaints (CC). In such a situation, a physician or provider has to maintain the documentation of all events and incidents, current treatments, allergies and review of systems (*RoS)* of a patient. A software application needs to be designed to ask hundreds of questions and thousands of care plans. The information is captured in attributes with given options to minimize error in accuracy. E.g. questions like "Severity of fall?", "How many times did you fall in last 30 days", "Did you suffer a fracture?", all these questions will be populated with predefined options. Also, these options are fired based on adaptive learnings.

3. Problem Statement

A conventional data storing and retrieval model for such a specific designed application could be challenging to performance or even degrade the efficiency of the response time. For applications with specific performance requirements, native key-value pairs provide a more lightweight and optimized solution compared to the overhead associated with parsing structured data formats.

So given R = { R_1, R_2, R_3, R_4……R_n}, where the *Rn* is unstructured set of questions and options (Called "Form") of a specific subject *p*. The End goal is to convert. each Ri into a set of {<$Key_{np,}$ $Value_{np}$ >} where each Key_{np} represents some a request and response (question and answer) of specific topics e.g. (physical exam, events, routine visits etc.)

4. The Model

The primary advantage of key-value pairs is the quick retrieval of data. By specifying the key, the associated value can be easily accessed. This enables efficient data lookup and retrieval operations. Keys must be unique within a given collection. This uniqueness ensures that each key corresponds to a single value, preventing ambiguity in data retrieval. Key-value pairs are used in database systems, and key-value stores for persistent data storage. They enable efficient indexing and retrieval of data. The simplicity of key-value pairs facilitates ease of implementation and integration into various programming environments.

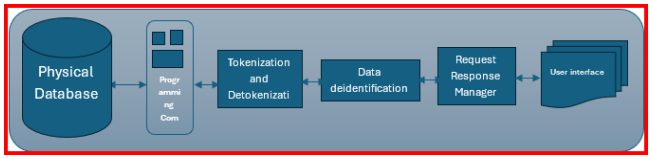

Figure 1: High Level design of Request Response model. The design does not show the logical boundaries, but the intent is to explain the functions of components

Physical database is the persistent store where Key value {<Key_{np}, $Value_{np}$ >} will be stored and retrieved when needed. The other aspects of physical database management are assumed to be part of physical database. **Programming components** are part of database management e.g. stored procedures, user defined functions, triggers etc. **Tokenization and detokenization** is the core part of application implementation, a set of design patterns, algorithms, data structures and control logics. This module is responsible for identifying the key, maintaining the uniqueness of the key, separating key and values on request, and associating them back on response. This module is the parser who parses the strings with predefined delimiters and joins back when required. Apart from this this module maintains the integrity of request and response, caching and ensuring the transaction is not broken. **Data deidentification** module identifies the data when in transit with cryptic algorithm so that health related information will be secure and compliant to the norm. **The response manager** retrieves the final tokenized/ detokenized data and prepares in format to consume by user interface. This is mainly an application layer protocol formatted response. A **user interface** is software application designed in any technology mainly web or windows application. Unlike JSON and XML, which are text-based, native key-value pair solutions are often binary, further improving efficiency in terms of both storage and transmission.

5. Request and response

The core design is called "Request Response". When the application calls for data saving module, the integral algorithm runs in iterative mode and captures all information available on interface. Later this module filters the information and extracts only required information which needs to be stored into a database. After the extraction of a string builder component builds the unique strings with key (element name) and actual data value, each key value pair is delimited with unique asci character. These delimiters are later used to parse the text string built by algorithm. Key-value pairs are well-suited for scenarios where quick data lookups and retrievals are essential, making them ideal for certain types of applications or databases.

ALGORITHM 1 1:

```
1. procedure GETKEYResponses:
2. Input: S= {<Key_np, Value_np >}:
3. Output: A unique key-value string with delimited characters
4:      Form_p ← {}
5:      Key_Labels ← {}
6:      Response_Values ← {}
7:      for each p in Form_p do
8:       form_ResponseString() = ""
9:       for each y in Key_Labels.Elements() do
10:            for each x in ValidResponse(y) do
11:                 ValidateResponse(x)
12:                 AppendResponse(x)
13:               DelimiteResponse(x,delimitChar)
14:           end do
15:         delimitChar = delimitChar++
16:         DelimiteResponse(x,delimitChar)
17:      end do
18:      print 'Form String:' form_ResponseString()
19:   end do
20:   return form_ResponseString()
```

Figure 2: Request Response algorithm.

The generic nature of this algorithm fits to any size of user interface, web, or windows. The interface must possess the property to hold the collection of elements / objects. The final output is prepared by the function $form_{responses}()$ which returns the character string, encrypted by de-identification component so that response should not be returned in human readable format or should not be intercepted when in transit. This ensures the HIPAA compliance policy.

6. Model Comparison

The following illustration explains the efficiency of request and response model. Consider patients being assessed and their reposes are recorded in application near to live time. The comparison study below explains the difference between the traditional saving and pre-population of data and "Request-response" model.

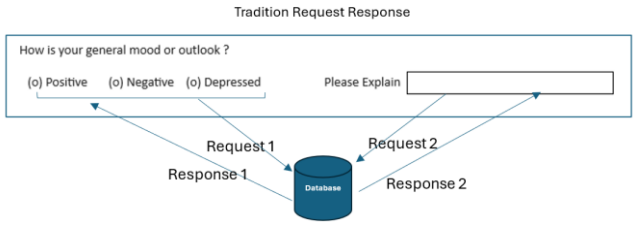

Figure 3: Traditional flow of Request Response mechanism. Each request is responded with the separate response, increasing the latency time and decreasing the response time.

In computing, latency refers to the time delay between the initiation of a request and the corresponding response or action. It is a crucial aspect of system performance and user experience.

The latency of the above traditional approach can be defined as below. However, there are several latency factors that play a vital role in Total Latency but for this context, we consider only required for processing the data, Disk, and IO latency factors.

Total Latency (L) = Processing Latency (PL) + Disk Latency (DL) + I/O Latency (IOL)

$$L = PL_{Req1} + PL_{Req2} + DL_{Req1} + DL_{Req2} + IOL_{Req1} + IOL_{Req2}$$

$$L = PL\ (Req1 + Req\ 2 + Req3\ldots\ldots Reqn) + DL\ (Req1 + Req\ 2 + Req3\ldots\ldots Reqn) + IOL\ (Req1 + Req\ 2 + Req3\ldots\ldots Reqn)$$

$$L = \sum_1^n PL + \sum_1^n DL + \sum_1^n IOL$$

The Total Latency L increases in proportion to the n, the higher the 'n', the more is the Latency Time L. The.

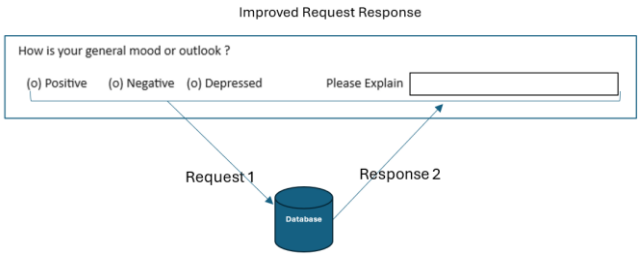

Figure 4: Improved flow of Request Response mechanism. Each logical request is bundled in single request and send for processing, reduces the IO latency and quicker response time

The 'Request Response' model tries to minimize the DL_n and IOL_n, however. PLn might be the same or less but not more. The reason behind this assumption is that this model expects the amount of time consumed by Processing latency (PL) for each separate request -the response attribute is the time required to build the Response string.

Here the volume of requests and response is reduced to half but in some cases, its may be reduced more, we are calling that reduction factor as 'j', where j <= n

So, in this case our latency time equation becomes.

$$L = \sum_1^n PL + \sum_1^{n-j} DL + \sum_1^{n-j} IOL$$

The success of this model and response time is merely depends on the "j" factor, as you see j <= n and n-j always less than n. So, the throughput of this model is always more efficient than the traditional model.

On the other side when the data is retrieved from the database upon receiving a pre-population request from the application, the exact reverse process is followed. The {<Key_{np}, $Value_{np}$ >} pair is extracted from the database and packaged in a response string, returned to the application. The response manager securely parses and returns the attributed data to the requested interface. The response manager also ensures the integrity of the transaction and establishes a continual relay of requests and response calls. Native key-value pairs enhance performance in scenarios requiring rapid data access, such as caching mechanisms or in-memory databases.

Some of the silent features of this model is,

1) Generic in nature, fits odd sizes and adapts browser or Windows-based applications.
2) Scalable -There is no limitation on how far it can go. However, the database data types can limit the boundaries for building and parsing the response string.
3) Improved Latency time- All requests are bundled in one package avoids server round trips and IO operations.
4) Improved efficiency through minimizing response time and network congestion
5) Less or nearly no development efforts are required. Easy to plug in and plug out.
6) Design standards and easy to follow. This model expects more design efforts than development efforts.
7) Flexible – The request response algorithm is easy to accommodate complex changes by adding iterative nested loops.

7. Conclusion

However, there is substantial growth in the latest computing infrastructure, the world is heavily dependent on text-based, heterogeneous forms of information exchange. With the increase of the latest trends in similar areas, a JSON implementation can strengthen the request-response model in the future time. However, the Request Response model explained here is at the core and leverages the native technology capability without any additional layer. Native key-value pair solutions explained here, provided by programming languages or databases, offer a more efficient and direct means of organizing and accessing data compared to JSON or XML formats. The simplicity of key-value pairs reduces the complexity of data representation, making it easier to manage and understand. JSON and XML are more verbose, leading to larger file sizes, whereas native key-value

pairs tend to be more concise, optimizing storage space. While JSON and XML are versatile and offer human-readable formats suitable for diverse data types, native key-value pairs excel in scenarios where speed, efficiency, and simplicity are paramount, offering a tailored solution for specific use cases.

Appendix

Latency - In computing, latency refers to the time delay between the initiation of a request and the corresponding response or action.

IO – Input Output

JSON – JavaScript Object Notation. It is a text-based open standard data interchange setup and only provides a data encoding specification.

XML- Extensible Markup Language. A markup language is a set of codes, or tags, that describes the text in a digital document.

References

[1] Aone, C.; Okurowski, M. E.; Gorlinsky, J.; and Larsen, B. 1999. A trainable summarizer with knowledge acquired from robust NLP techniques. 71–80.
[2] Sha, F., and Pereira, F. 2003. Shallow parsing with conditional random fields. NAACL ’03, 134–141
[3] C.L. Philip Chen, Chun-Young Zhang “*Data-intensive applications, challenges, techniques and technologies: A survey on Big Data*” vol 275, Information Science
[4] Pradhan, S.; Hacioglu, K.; Ward, W.; Martin, J. H.; and Jurafsky, D. 2003. Semantic role parsing: Adding semantic structure to unstructured text. In Proceedings of the Third IEEE International Conference on Data Mining, ICDM ’03, 629–. Washington, DC, USA: IEEE Computer Society.
[5] Zhang C, Ming Y, Wang M, Guo Y and Jia X. Encrypted and Compressed Key-Value Store With Pattern-Analysis Security in Cloud Systems. IEEE Transactions on Information Forensics and Security. 10.1109/TIFS.2023.3320612. 19. (221-234).

Author Profile

Santosh S Deshmukh. The author is a seasoned IT professional, with an illustrative career of over two decades in Information Technology (IT) and a profound wealth of knowledge in the US healthcare landscape. The author is an independent researcher, specialized in developing cutting-edge products, security implementation and large-scale health systems. The Author is 48 years old and a current resident of McKinney, North Texas. The Author earned a master’s degree in Business Analytics from Grand Canyon University, Phoenix, AZ in 2020. His fields of study are Computer science, Business Analytics and Healthcare.

He worked as Sr. PROJECT CONSULTANT (IT) for esteemed organizations like United Health, Blue Cross Blue Shield, CVS, Emblem Health, and General Motors. Currently, he has been working as a Project Consultant for Anthem Inc. since 2019. For the past 15 years, he has been at the forefront of managing complex healthcare IT initiatives, demonstrating a keen understanding of the dynamic and regulated healthcare environment. With a significant focus on Healthcare Analytics, he has seamlessly integrated his knowledge to address the unique challenges within the healthcare sector, contributing to the improvement of patient outcomes and operational efficiency.

Mr. Deshmukh is a certified Project Management Professional (PMP) since 2009. Mr. Deshmukh is an expert professional with a unique blend of technical acumen, project management expertise, and a significant impact on the U.S. healthcare system. Besides, Mr. Deshmukh is also a lifetime member of Sigma Beta Delta (SBD). An honorary member of IEEE since 2020 and Sr. Member since 2021. Mr. Deshmukh is also a member of the Advisory Board of Our Lady of the Lake University, Houston. Mr. Deshmukh served in a secretary position at a non-profit organization during 2022-23. Author's email id is san_desh3@yahoo.com