Hybrid LRU Algorithm for Enterprise Data Hub using Serverless Architecture

Dr. Murugan A – Associate Professor and Ganesan S – Research Scholar,
PG & Research Dept. of Computer Science, Dr. Ambedkar Govt. Arts College,
University of Madras, Chennai, India

Article History: Received:11 January 2021; Accepted: 27 February 2021; Published online: 5 April 2021

Abstract— In [1], hybrid LRU algorithm was built to execute parameterized priority queue using Least Recently Used model. It helped to determine the object in an optimum mode to remove from cache. Experiment results demonstrated ~30% decrease of the execution time to extract data from cache store during object cache extraction process. In the era of modern utility computing theory, Serverless architecture is the cloud platform concept to hide the server usage from the development community and runs the code on-demand basis. This paper provides Hybrid LRU algorithm by leveraging Serverless Architecture benefits. Eventually, this new technique added few advantages like infrastructure instance scalability, no server management, reduced cost on efficient usage, etc. This paper depicts about the experimental advantage of Hybrid LRU execution time optimization using Serverless architecture.

Index Terms— Serverless, Utility computing, Big Data, Scaling, Metered payment, Enterprise Data Hub, etc.

I. INTRODUCTION

Data is foundation of the computer industry. Microsoft's Bill Gates quoted that Data is the biggest asset in the modern world. In the rapid transformation of Information Technology, enterprise data growth [15] is phenomenal. Enterprise Data Hub [1] is a solution to build and maintain the golden records of any enterprise as shared trustable enterprise data. With the big data Map Reduce algorithm, EDH is easily built using the traditional computing model. In recent industry days, serverless computing is highly adopted due to favorable small, self-contained units of computation, which makes the big data process easier to manage and scale in the cloud. Serverless computing programming model is highly influenced with inherit model of non-maintenance state of the underlying application.

This paper depicts about the efficient way of utility computing to handle the data in the most effective logic. In the recent social media world, data loads continue to grow exponent. It is highly expected to see the serverless concepts as a standard approach for offloading functions to execute closer to end users and devices. By reducing costs, latency, time to market, and complexity, the serverless model is poised to become a staple of the app space. Key logic of this paper, is to build the serverless computing for EDH design.

II. LITERATURE REVIEW

A. *Caching Algorithms*

Computer Industry evolves with multiple Cache replacement algorithms [8] as below:

1. Optimal Cache algorithm (OPT)
2. Least Recently Used (LRU)
3. Least Frequently Used (LFU)
4. Adaptive Replacement Cache (ARC)
5. Low Inter-reference Recency Set (LIRS)

Implementation depicts about the experimental advantage of execution time optimization and efficient page/cache hit ratio, using hybrid LRU algorithm with the optimal combination of LRU and priority models. The placement of data within the memory system is crucial to reducing the number of conflict misses to the minimum possible.

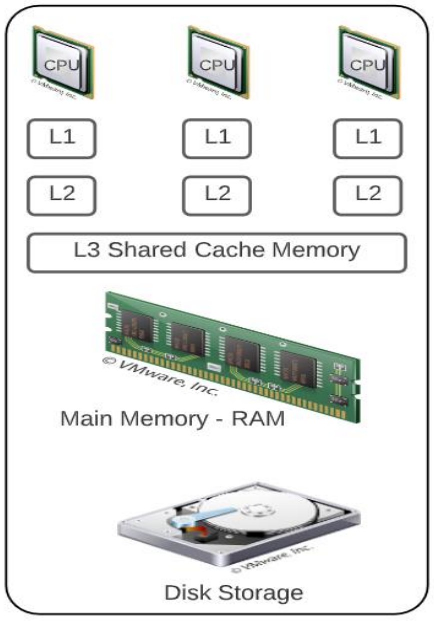

Fig. 1. System memory models

Computer system memory models are depicted in the above diagram with the different levels of efficiency and persistence.

B. Enterprise Data Hub

Enterprise Data Hub is a solution to build and maintain the golden records of any enterprise as shared trustable enterprise data. It is the proposed solution which is a central data repository for any enterprise with open sourced Big Data tools and techniques like Hadoop Distribution File System, Map Reduce, Hive, etc.

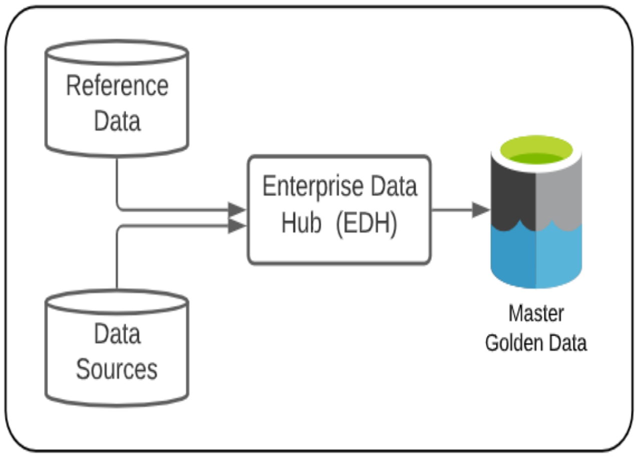

Fig. 2. ReferenceArchitecture of EDH

The target output of EDH is the golden master data for an enterprise.

C. Shift in Serverless Architecture

In recent times, an architecture model is emerged with the name - ‘Serverless’. In reality, this term doesn’t literally mean as ‘no server’. Function as a Service (FaaS) concept emerges along with this new architecture. It allows small pieces of code represented as functions to run for limited amount of time on demand basis in the cloud.

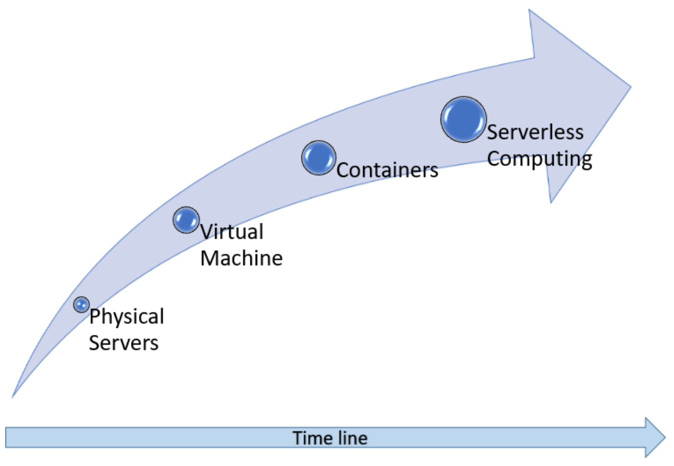

Fig. 3. Architecture Evolution of Serverless Computing

As mentioned above, the traditional computing models took months to deploy any system in production with the life cycle in years. Serverless Architecture is disruptive design pattern to support the modern utility computing. On evolution of VM and containers concepts, serverless computation deploy the production ready code with self-contained small units in the cloud.

III. Serverless Architecture

A. Industry Use Case

Allied Market Research report estimated the global serverless architecture market is worth around $7.6 billion in 2020, and forecasts to ~$22 billion by 2025 with annual growth rate of 28%.

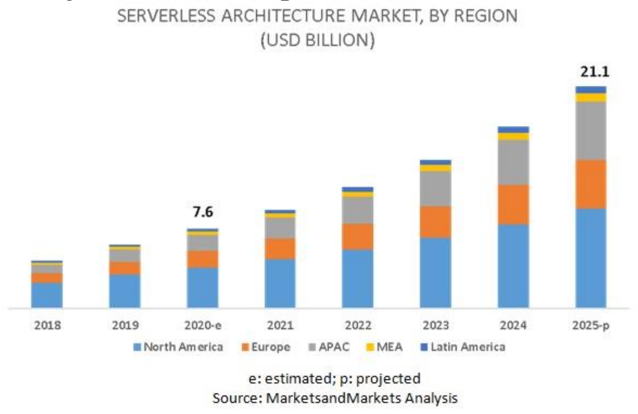

Fig. 4. Serverless Architecture Market by Region

This design helps to parallelize into independent units of work in asynchronous and concurrent mode without worrying about the underlying infrastructure. In terms of changing business requirements, it is highly dynamic for accelerated developer velocity.

B. Message Driven Design - Illustration

Message driven design is the best suit for the serverless computing. In theory of computer architecture, message driven design is built based on the asynchronous communication. A message is a simple data transfer object (DTO) with message name and details. On arrival/dispatch of any message, the system triggers a function to execute. It is the fundamental of message driven application.

The context is usage of message driven design in FaaS. Let us consider a real life application – Advertisement (Ad) server. Traditionally, Ad Server is designed synchronously with the response to the user click operation in a channel. In terms of operation, the user clicks on the advertisement content in the browser and the server collects the relevant information for further processing.

It is redesigned with message consumer model using FaaS function of asynchronous message processing is a very popular use case for serverless architecture. This function runs within the event-driven context the vendor provides. FaaS is distinct to process several messages in parallel by instantiating multiple copies of the function code. Programmer doesn't need to worry about the underlying infrastructure as it is taken care by serverless technologies.

C. Benefits of Serverless computing

In term of the business benefits, Serverless computing provides in four categories namely Scalability, Simplified server execution, Time to market and Cost advantage. With the power of scaling demand seamless and simplified abstracted server execution, the product reach the market on time.

Serverless architecture makes the availability of the program instantly and so it significantly reduces the software deployment cycle. Serverless computing requires only the operational cost, no upfront capital. The rate of cost advantage is depicted as below for the serverless computing.

D. Limitation of Serverless computing

By design, serverless computing is intentionally ephemeral. As the execution is sustained for few minutes in the cloud, it doesn't fit for long running tasks. Also, it doesn't retain any stateful data of the previous run.

IV. Technical Implementation

A. Priority Queue Design

Computing theory depicts the impact of using different priority queues in the system performance. The core concept is by having significant overhead in the constant factor and so it performs the best. Priority Queue performance [4] metric is

Table- I: Priority Queue Processing Time

#	**Priority Queue Model**	**Insert/Delete(min)**
1	Sequence Heap (Cache aware)	$O\ (log_4 N)$
2	Bottom up Binary Heap (worst case)	$O\ (log_2 N)$
3	Aligned 4-ary Heap (worst case)	$O\left(\frac{1}{B}\ log_k\ \frac{N}{l} + \frac{1}{k} + \frac{log_k}{l}\right)$

N:queue size, B: block size, M: cache size, l: θ(M), k: θ(M/B)

During this traversal, node flag is set to denote the direction, which is opposite to the taken direction.

B. Hybrid LRU Algorithm

The objective is to build the hybrid LRU with the combination of priority queue and LRU concepts. An improvement of the existing LRU is termed as Hybrid LRU, in which parameterized priority queue logic is extended to obtain the execution performance benefits.

On comparison with the normal queue concepts, Priority Queue has the below properties.

1. Priority is associated with each item.
2. Higher priority element is dequeued before the lower priority element.
3. In the event of the same priority, their queue order will take the priority.

A typical priority queue supports three operations. First operation named push is to insert an item with given priority. Second operation pop is to fetch and remove the highest priority item. Last method peek is to get the highest priority element in the queue without removing it.

Pseudo code of the logic will be:

```
------------------------------------------------------------------
Algorithm: Priority queue algorithm to maintain
Input: Gets the content or position
Output: Retrieves the content for the given position
```

```
--------------------------------------------------------------------
class NodeObject
begin
   Declare dataStore as integer
   Declare priorityNum as integer
   Declare next as NodeObject
end

function newNode (d integer)
begin
   Initiate temp as NodeObject
   Assign parameter d into temp.dataStore;
   Assign parameter d into temp.priorityNum;
   Assign null to temp.next;
   Return temp;
end

function push (
      head NodeObject, d integer, p integer)
begin
   Assign start as head;
   Assign temp as newNodeObject(d, p);
   if  ((head).priorityNum is greater than p)
   begin
      temp.next = head;
      (head) = temp;
   end
   else
   begin
      While (start.next != null &&
         start.next.priorityNum < p)
      begin
         Move start into start.next;
      end
      Assign temp.next as start.next;
      Assign start.next as temp;
   end
   return head;
end
```

Key differentiator of a priority queue is processing by the given priority instead of traditional first in first out model. In terms of data structure, it is an abstract data type to capture the idea of a container, in which elements are attached with priorities value. In essence, an element with highest priority is positioned at front of the queue. On removal of the element, it picks up the next highest priority element to the front.

In terms of technical implementation, LRU element is identified with the combined logic of parameterized priority queue element.

C. Serverless Architecture

In the process of architecture evolution, Serverless computing [6], [7] is primarily associated with two key concepts. They are (1) Applications that rely on third-party cloud services to handle their business logic and state, which is commonly called as Background-as-a-Service (BaaS) and, (2) Applications in which the server side logic is written by application developers. Unlike traditional architecture, the differential element is to execute in stateless containers that are event-triggered and fully managed by a third party system. It is termed as Functions-as-a-Service (FaaS).

Serverless FaaS model is designed to implement the application logic as functions, which may be written in various languages and exposed as web services, and packed with their dependencies.

Pseudo code is drafted for hybrid LRU algorithm using serverless redesign as below:

```
-------------------------------------------------------------------
Algorithm: Hybrid LRU algorithm using serverless
Input: Gets the content or position
Output: Retrieves the content for the given position
-------------------------------------------------------------------

class ServerlessHybridLRU implements RequestHandler
begin

   class NodeObject
   begin
      Declare dataStore as integer
      Declare priorityNum as integer
      Declare next as NodeObject
   end

   function push (
                  head NodeObject, d integer,
                  p integer, Context context)
   begin
      Assign start as head;
      Assign temp as newNodeObject(d, p);
      if  ((head).priorityNum is greater than p)
      begin
         temp.next = head;
         (head) = temp;
      end
      else
      begin
         while (
               start.next != null &&
               start.next.priorityNum < p)

         begin
            Move start into start.next;
         end
         Assign temp.next as start.next;
         Assign start.next as temp;
      end
      context = this;
      return head;
   end

end
```

Apart from on demand cloud infrastructure, there are two key improvements in application objects coding, namely event and context. Serverless design event differs in structure and contents, depending on which event source created it. In our algorithm, the event will be triggered during the priority queue push operation. Actually, the contents of the event parameter, include all of the data and metadata for the given push FaaS function to drive the serverless design. The context object allows function as a code to interact with the serverless execution environment. The contents and structure of the context object vary, based on the runtime language at minimum.

During the execution of ServerlessHybridLRU class object, each time a function container is created and invoked. In the system, it remains active and available for subsequent invocations for at least a few threshold minutes before it is terminated. On the subsequent invocations occur on a container that has already been active and invoked at least once before, then invocation is running on a warm container. When an invocation occurs that requires function as a code package to be created and invoked for the first time, the invocation is experiencing a cold start by design.

When RequestHandler logic is executing using the given serverless context, the system takes the advantage of a warm container. As a result, the serverless design creates faster code execution for the associated Hybrid LRU algorithm. In turn, the experiment results are in quicker responses and lower cost when the input data set grows from 10 to 10,500 records as depicted in the following experimentation result section.

V. EXPERIMENTATION RESULT

A. Data Analysis

To experiment the value of serverless architecture on applying in hybrid LRU, the previous data set was executed with multiple cycle between traditional and serverless architecture. Amazon Lambda provides the cloud platform for the execution of high throughput job. Each serverless FaaS function provides the service of memory between 128 to 3,008 MB, disk storage of 75 GB, concurrent execution of 1,000 tasks and timeout of 15 minutes.

With the above obtained results, corroborate the benefits of the serverless architecture in three core areas. They are (i) drastically reducing the burden of infrastructure management, (ii) allowing more functionality to be deployed on fog nodes with limited resources, and (iii) fulfilling the requirements of different application scenarios and heterogeneous deployments of fog nodes. In the essence, the execution time of Hybrid LRU is optimized when the computing process grows.

B. Execution Time Optimization

Serverless hybrid LRU algorithm is benchmarked to measure the response time with meaningful experimental executions. Serverlesss Function as a Service (FaaS) is executed against the traditional design of hybrid LRU with four handful use cases.

Table- II: Experimented Result – Execution Time

Patterns - architecture	**Data set execution (milli-seconds)**			
	10	**150**	**1,300**	**10,500**
Traditional HybridLRU	2.1	3	6.4	12.5
Serverless HybridLRU	2.1	3	6.1	11.2

The execution data set contains the range from 10 to 10,500 and approximately 6.39 GB sized content. Python 2.7 and Node.js software are used to develop the program. The system uses Amazon cloud's Lambda to leverage the serverless architecture. The four queries are tested and execution time is captured against disk based hive and serverless map reduce algorithms.

Serverless Execution Efficiency α Transaction Volume

Based on the experimental results, it is evident that the execution efficiency using serverless architecture is directly proportional to the given volume of transactions. Because, serverless design is good fit for the compute intensive programs to achieve the efficiency.

Experimented execution results are marked in the above table to demonstrate the time efficiency. Data points are graphically represented as below.

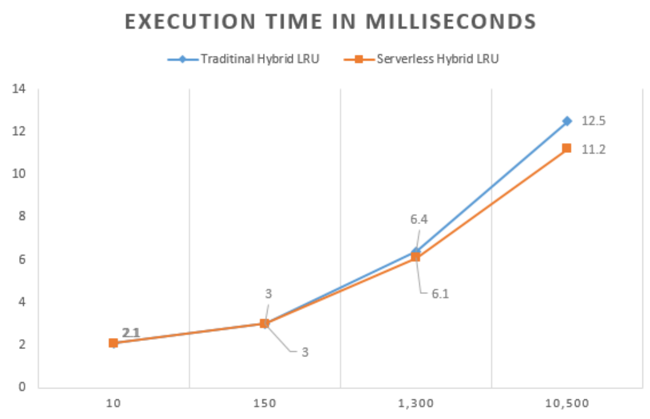

Fig. 5. Comparision chart of Scan & Aggregation queries

Result shows the serverless architecture improvement of hybrid LRU algorithm execution time between traditional and serverless models. When the rate of data volume and complexity of queries increase, the execution time efficiency is improved as shown in the experimented result graph.

VI. Conclusion

This paper describes an improved version of hybrid LRU algorithm, by leveraging the emerging serverless architecture. Amazon Lambda framework is used to implement serverless architecture in this paper. The core logic is to invoke Function as a Service in response to the distributed data based on the number of simultaneous queue function. On comparing the performance of serverless architecture against traditional design, it has an impact on the execution time efficiency of data queue processing.

This research paper concludes the technical strength of new serverless architecture, using the experimental results. Thus, enterprise data hub can leverage the state of the most efficient architecture to execute the newly built hybrid Least Recently Used data processing algorithm.

References

[1] Murugan A. and Ganesan S., “Hybrid LRU Algorithm for Enterprise Data Hub”, International Journal of Innovative Technology and Exploring Engineering, ISSN: 2278-3075, Volume-9, Issue-1, November 2019
[2] B. Sai Jyothi & S. Jyothi, “A Study on Big Data Modeling Techniques”, International Journal of Computer Networking,Wireless and Mobile Communications (IJCNWMC), Vol. 5, Issue 6,pp, 19-26
[3] Bernstein, D. Containers and cloud: From LXC to Docker to Kubernetes. IEEE Cloud Computing 1, 3 (Sept. 2014), 8184
[4] A. Kaleel Ahmed, C. B. Senthil Kumar & S. Nallusamy, “Study on Environmental Impact Through Analysis of Big Data for Sustainable and Green Supply Chain Management”, International Journal of Mechanical and Production Engineering Research and Development (IJMPERD), Vol. 8, Issue 1, pp, 1245-1254
[5] Baldini, I. et al. Serverless computing: Current trends and open problems. Research Advances in Cloud Computing, Springer, 2017, 120
[6] Udayakumari Vidhyasagara Menon & Muhammed Refeque, “Big Data Analytics: Implications on Economic Planning and Implementation”, International Journal of Business and General Management (IJBGM), Vol. 5, Issue 5, pp; 59-64
[7] CNCF Serverless White Paper; https://github.com/cncf/wg-serverless#whitepaper
[8] G. Kumaresan & P. Rajakumar, “Predictive Analytics Using Big Data: A Survey”, BEST: International Journal of Management, Information Technology and Engineering (BEST: IJMITE), Vol. 3, Issue 9, pp, 61-68
[9] Yan, M., Castro, P., Cheng, P., Ishakian, V. Building a chatbot with serverless computing. In Proceedings of the 1st Intern. Workshop on Mashups of Things, 2016
[10] M. Padmavathi & Shaik Mahaboob Basha,” A Conceptual Analysis on Cloud Computing ACO Load Balancing Techniques”, International Journal of Computer Science and Engineering (IJCSE), Vol. 6, Issue 4, pp; 39-48
[11] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara, “Serverless computing: An investigation of factors influencing microservice performance,” in 2018 IEEE International Conference on Cloud Engineering, IC2E 2018, Orlando, FL, USA, April 17-20, 2018, 2018, pp. 159–169.

[12] Jayalakshmi K, Uma K M, Veena A, & Lavanya Santhosh, “A Quantitative Analysis of Security Issues in Cloud Computing”, IMPACT: International Journal of Research in Engineering & Technology (IMPACT: IJRET), Vol. 5, Issue 8,pp, 51-60
[13] M. Roberts, “Serverless architectures,” https://martinfowler.com/articles/serverless.html, March 2019.
[14] Ye, W., Khan, A.I. and Kendall, E.A. Distributed network file storage for a serverless (P2P) network. In Proceedings of the 11th IEEE Intern. Conf. on Networks, 2003, 343347
[15] Wang, L., Li, M., Zhang, Y., Ristenpart, T. and Swift, M. Peeking behind the curtains of serverless platforms. In Proceedings of USENIX Annual Technical Conf., 2018, 133146. USENIX Association
[16] Lin, W-T, Krintz, C., Wolski, R., Zhang, M., Cai, X., Li, T. and Xu, W. Tracking causal order in AWS lambda applications. In Proceedings of the IEEE Intern. Conf. on Cloud Engineering, 2018
[17] Baldini, I., Castro, P., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Suter, P. Cloud-native, event-based programming for mobile applications. In Proceedings of the Intern. Conf. on Mobile Software Engineering and Systems, 2016, 287288. ACM, New York, NY.
[18] Sam Newman. 2015. Building Microservices (1st ed.). O’Reilly Media, Inc.
[19] Etzioni, O. and Niblett, P. Event Processing in Action. Manning Publications Co., Greenwich, CT, 2010
[20] Lee, H., Satyam, K. and Fox, G.C. Evaluation of production serverless computing environments. In Proceedings of IEEE Cloud Conf. Workshop on Serverless Computing (San Francisco, CA, 2018)