

# DLB: Deep Learning Based Load Balancing

Xiaoke Zhu<sup>§\*</sup>, Qi Zhang<sup>†\*</sup>, Taining Cheng<sup>§</sup>, Ling Liu<sup>‡</sup>, WeiZhou<sup>§\*\*</sup> and Jing He<sup>§\*\*</sup>  
 Yunnan University<sup>§</sup>, IBM Thomas J. Watson Research<sup>†</sup>, Georgia Institute of Technology<sup>‡</sup>

**Abstract**—In this paper, we introduce DLB, a Deep Learning based load Balancing mechanism, to effectively address the data skew problem. The key idea of DLB is to replace hash functions in the load balancing mechanisms with deep learning models, which are trained to be able to map different distributions of workloads and data to the servers in a uniform manner. We implemented DLB and deployed it on a practical Cloud environment using CloudSim. Experimental results using both synthetic and real-world data sets show that compared with traditional hash function based load balancing methods, DLB is able to achieve more balanced mappings, especially when the workload is highly skewed.

**Index Terms**—load balancing, consistent hashing, neural networks, cloudsim

## I. INTRODUCTION

With the development of Cloud computing, companies are becoming increasingly interested in migrating their services and data to Cloud platforms, such as AWS [1], IBM Cloud [2], Google Cloud [3], and Microsoft Azure [4], on which the effectiveness of load balancing is of great importance. Maintaining a balanced workloads benefits the cloud service provider by not only increasing the utilization of their resources, but also improving the quality of services. Currently, hash function based load balancing mechanisms are the dominant design, in which hash function based approaches are used to determine which server a workload needs to be assigned to.

A hash function is able to generate balanced results when the input data is uniformly distributed. However, the real-world data sets often exhibit remarkable skew. For instance, analysis of air traffics and online human behavior data sets [5], [6] revealed that such data usually follows different power law distributions. When the input data is skewed, the output of the hash function will also be skewed. Therefore, using a hash function in a load balancing mechanism can result in unbalanced workloads assignment when data skew exists in the input. Even worse, such unbalanced workloads could seriously harm the performance of applications and services running on distributed platforms. Elaheh Gavagsaz and et al. [7] demonstrated that traditional join algorithms based on MapReduce are not efficient when working with skew data, Joanna Berlinska and et al. [8] also revealed that the uneven distribution of the keys might cause imbalance computation

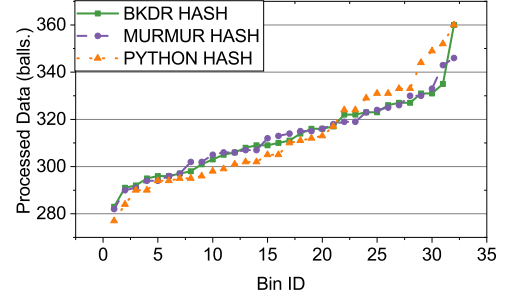


Fig. 1: Data skew for different hash functions

completion time among different MapReduce tasks, which eventually prolonged execution of the whole MapReduce job.

There are several reasons why hash functions do not perform well on skewed data sets. First, the hash function was originally designed to perform fast index [9] (i.e., indexing with  $O(1)$  time complexity), compression [10] (i.e., compressing a large input in a deterministic way), cryptography [11] (i.e., irreversible mapping from inputs to outputs) and etc., thus dealing with skewed data was not considered as one of its primary design goals. Second, although there have been efforts, such as BKDR hash [12], MURMUR hash [13], and Python hash [14], to enhance the hash functions to better handle the skewed data, their effectiveness are not satisfying. As shown in Figure 1, we manually generate a skew data set under normal distribution as the inputs and use the above mentioned three hash functions to map these inputs to 32 bins. After that, the bins are sorted by the number of inputs assigned to it. Ideally, the lines in this figure should be flat, which means different bins are taking a similar amount of inputs if the hash functions are able to map the input to a uniform distribution. However, the lines in the figure are all in an increasing trend. The numbers of inputs being assigned to different bins vary from 280 to 360, which are significantly unbalanced.

The availability of big data and the rapid advance of AI techniques provide unique opportunities to rethink the design of load balancing mechanisms by making them perform better on skew data. The key idea is as follows: instead of using a hash function, learned models can be applied to determine where the inputs should be mapped to the hash circle. Although model training is required beforehand, this approach is practical and has several advantages compared to traditional hash based methods. First, the capability and affordability of collecting large amount of data nowadays

This work was supported in part by the National Natural Science Foundation of China under Grant 61762089, Grant 61663047, Grant 61863036, Grant 61762092 and Open Foundation of Key Laboratory in Software Engineering of Yunnan Province under Grand 2020SE310

\* made equal contribution to this work.

\*\* to whom correspondence should address {zwei,hejing}@ynu.edu.cn

provides the potential to train data distribution aware models for load balancing mechanisms. Second, the distribution of data collected by a specific company or organization for a given task is usually consistent, which is demonstrated by analysis results from [5], [6], [15]. This shows the feasibility of using historical information to deal with future workloads. Third, when appropriately trained, the output of a learned model can be uniformly distributed even when the input data set is highly skewed.

In this paper, we propose DLB, which uses deep learning models to effectively address the data skew problem in existing load balancing mechanisms. Researchers [16]–[19] have explored the possibility of partially replacing existing data structures and algorithms with deep learning models. For example, Tim Kraska and et al. [16] introduced the hash model index, which reduces the total number of hash conflicts over map data set by learning a CDF(Cumulative distribution function) at a reasonable cost. However, there are still remaining challenges to leverage deep learning models to improve the effectiveness of load balancing mechanisms. On the one hand, how to design a neural network that can converge quickly during the training while also being able to effectively mapping large volumes of inputs to a uniformly distributed space. On the other hand, how to balance between the complexity and the expressiveness of the model. Concretely speaking, a simple neural network can be easily trained, but it will not be able to map large amount of inputs into a uniformly distributed space without incurring significant conflicts. While a complex model can reduce the mapping conflicts, but it cannot be trained easily due to gradient dissipation and explosion problems.

In order to solve these challenges, DLB is designed in a way that, instead of using a single end-to-end model, it organizes a set of models into a hierarchical architecture. In such an architecture, the models are organized in different connected layers. For a specific input, it will go through one model in each layer, while the model in the previous layer specifies which model in the next layer needs to be invoked. The final output will be the position on the hash circle for this input. Since the distribution of input data is not guaranteed to stay the same, DLB also continuously monitors the actual load distribution of all the servers to make sure no server becomes a hotspot. Compared with traditional hash function based load balancing mechanisms, such as Consistent Hashing [20] and Consistent Hash with Bound Load [21], DLB is able to map the input data sets to a uniformly distributed space even when they are highly skewed. In addition, compared with a single but complex end-to-end model, a hierarchical design makes each model converge more quickly during the training.

The main contributions of this paper are as follows:

- We designed DLB, a deep learning based load balancing mechanism which solves the data skew problem by replacing the hash function with deep learning models.
- We implemented DLB and deployed it in a practical environment using CloudSim [22], which enables

modeling and simulation of real Cloud computing systems and application provisioning environments.

- The effectiveness of DLB is measured by using both synthetic and real-world data sets under different distributions.

## II. RELATED WORK

Load balancing mechanisms are widely used in distributed computing environment to balance the workloads among different servers, and the effectiveness of such mechanisms is critical to the overall performance and service quality of the distributed platforms. Therefore, how to design an effective load balancing mechanism has attracted the interest of many researchers. In this section, we introduce related researches in this area, while at the same time, we also discuss the existing efforts on trying to use neural network based learned data structures to improve the performance of traditional systems.

**Hashing based load balancing.** As one of the mainstream load balancing mechanisms, Consistent Hashing(CH) [23] proposed by Karger and et al. has been widely adopted. Ideally, by using a randomized hash function, both balls and bins can be assigned to the hash circle in a uniformed way, so that different bins will be able to hold the similar number of balls. However, it is usually not the case in the real-world due to the existence of data skew in the inputs. There have been many efforts to address this issue [24]–[27]. For example, David R. Karger and et al. [20] tried to enable CH to generate more balanced results by using virtual bins which are replicas of real bins in hash space, and one real host can be correspondent to several virtual bins. The authors showed that the overall load balancing performance could be improved accordingly. To further address the data skew problem in load balancing, Johan Lamping and Eric Veatch proposed jump Consistent Hashing [25], which works by computing when its output changes as the number of bins varies. In this approach, the hash value of a ball is not randomly generated, but acquired according to the probability determined by the number of existing bins. Also, whenever a new ball is added, the hash value of the existing balls needs to be recomputed according to the a predefined probability. Roberto Grossi and et al. designed Round-Hashing [26]. Thaler and Ravishankar proposed [24] Rendezvous hashing algorithm, for a given ball  $q$  and  $n$  bins, it applies a hash function to the all the pairs  $\{q, p_i\}$ , in which  $i \in \{1 \dots n\}$ , and assigns the ball to the bin that can lead to largest hashing result.

**Neural network based learned data structures.** This thread of research explores the potential of utilizing the neural network based learned data structures to improve the performance of traditional systems [16], [17], [19], [28]. Among them, Tim Kraska and et al. proposed learned B-tree, learned hash, and learned bloom filter structure [16] to improve the indexing performance of traditional structure by learning the distribution from the historical data. Xiang and et al. [17] proposed a LSTM-based inverted index structure.

Different from the above-mentioned researches, instead of improving the effectiveness of traditional hash functions,

our DLB approach takes the advantages of both Consistent Hashing and deep neural network. In DLB, a deep learning model based on the historical data is trained, and then used to map the newly coming data to a uniformly distributed space even when such data is skewed.

### III. DLB: DEEP LEARNING BASED LOAD BALANCING

In this section, we discuss the details of DLB design. The goal of load balancing is to uniformly distribute different workloads on multiple servers so that no server will become the hotspot. A hash function, being the core building block in most of the existing load balancing mechanisms, can be considered as a black box that takes an input and maps it to a position on the hash circle. Therefore, we propose a Deep Learning based load Balancing mechanism named DLB, which replaces the hash functions with deep learning models to fulfill the same mapping task. We observe that this approach is able to work well on skew data and provide more balanced workload distribution compared with traditional hash function based load balancing mechanisms.

#### A. Design

1) *Hierarchical models*: As discussed earlier, a hash function in a load balancing mechanism can be replaced by a deep learning model, and a well trained model can generate uniformly distributed outputs even when the inputs are highly skewed. A natural question to ask is which model should be used. In load balancing mechanisms such as CH, the inputs are usually mapped into a large space (e.g.,  $2^{32}$ ) to avoid conflicts. If we consider each slot in this output space as a class, what the model needs to achieve is actually classifying each input into one of these different classes. This actually turns the mapping task into a classification task. Since the space of this classification is so large, training a single model for such a task will be really difficult.

Therefore, we propose an architecture of hierarchical models in DLB to address this problem. As shown in Figure 2, instead of using one single model, multiple models are involved to solve this classification problem. The models are organized into a hierarchical structure with different connected layers. To find out the position that an input should be mapped to on the hash circle, each input will need to go through a model in each layer. The whole mapping procedure can be divided into 4 stages, and details of each stage are described as follows:

**Input stage.** Various formatted features can be observed as the inputs of load balancing mechanisms when a hash function is used, being it an ID string of a user or a MD5 value of a file. However, these features need to be converted so that they can be consumed by a neural network model. Therefore, the goal of this stage is to pre-process the input data, such as converting strings or numerical data into vectors, so that they can be directly used as inputs of a neural network model.

**Disperse stage.** The main strategy used in this stage is *divide and conquer*. Concretely speaking, the disperse stage consists of multiple models which are organized in a

hierarchical architecture(i.e., a tree structure). All the models in this architecture work collaboratively to figure out which position on the hash circle a given input should be placed. Since the space of the final hash circle is usually very large, the motivation of this design is to *divide* a complex classification task, which is supposed to deal with a large output space, into multiple smaller tasks. In this way, the original complex classification problem can be *conquered* more effectively by solving these smaller problems. In other words, with such a hierarchical architecture, each model only needs to tackle a much simpler classification problem with only a subset of the whole output space. As shown in Figure 2, the models in this stage are split into multiple layers. The root model(i.e. the one on the left most in Figure 2) takes the input data set and determines which model in the next layer needs to be invoked, while the models in the other layers of the disperse stage go through the same procedure using their corresponding assigned input.

**Mapping stage.** Models involved in this stage are located in the last layer of the hierarchical architecture. Different from the models in the disperse stage that select which model in the next layer needs to be used, each model in the mapping stage is responsible for generating the position of a given input in its sub-circle. Different models in the mapping stage are correspondent to different sub-circles, which are actually areas on the hash circle, while they collectively cover the whole hash circle.

**Join stage.** Since the output of each model in the mapping stage is a position on each model's own sub-circle, another layer is needed to translate such a local position on a sub-circle into a global position on the hash circle. In order to create the continue hash circle, sub-circle of different mapping stage models are connected sequentially, thus the final global position  $Pos_{q_i}$  of input  $q_i$  on the hash circle is established by Eq 1.

$$Pos_{q_i} = \mu_i + (model\_id - 1)t \quad (1)$$

$model\_id$  is the ID of the model in the mapping stage starting from 1.

2) *Server Management*: In traditional load balancing mechanisms such as CH, both workloads and servers are mapped to the hash circle by hash functions, and a workload is assigned to its clockwise closest server. In DLB, a deep learning model is used to map a workload to a position on the hash circle, while a deterministic approach is used to map the servers. Although server mapping can also be done by using learned models, it is not necessary since the number of servers is usually much smaller than that of the workloads and a deterministic approach is good enough to evenly map the servers to the hash circle.

Similar to traditional load balancing mechanisms, DLB assigns a workload to a server in a clockwise manner. Since DLB is able to uniformly map the workloads to the hash circle, using a deterministic server mapping approach can achieve well balanced workload distribution. Concretely speaking, when a new server is added, DLB will add the server to

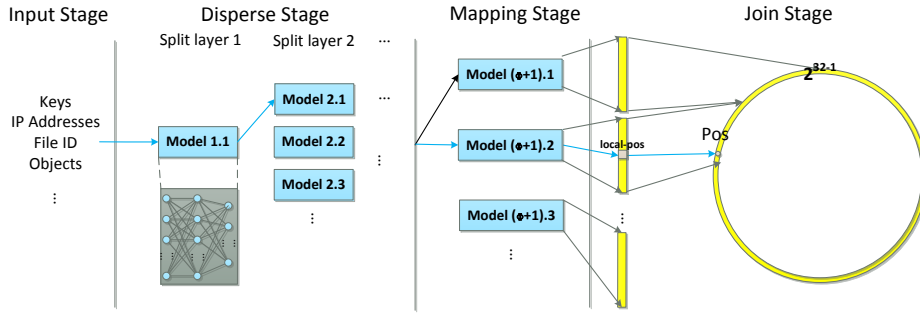


Fig. 2: The hierarchical model architecture in DLB

a place such that this server can evenly divided the largest sub-circle on the hash circle. When an existing server becomes unavailable, the workloads on this server will be reassigned to its clockwise next server. Similar to Consistent Hashing with bounded load(CHBL), each server in DLB has a load threshold  $\epsilon$ . A new workload can be assigned to a server only if the load of this server will not exceed  $\epsilon$ . Otherwise, other servers need to be considered.

### B. Training

In this sub-section, we discuss the considerations of how to train the hierarchical models mentioned above from two aspects.

First, how to label the training data. Given historical cluster access data, to make sure that the models will not generate skew output when the training data is skewed. Meanwhile, Since the hierarchical architecture includes multiple models in different layers, for each input, it needs to be labeled for each model. We discuss the labeling process for DLB from two aspects: creating labels used in the mapping stage models as well as in the disperse stage models. The difference between these two types of labels is that the former one represents positions on a hash circle, while the latter one is correspondent to the ID of the model in the next layer.

The method to create labels for DLB is depicted in Algorithm 1.  $tag^\phi(k_i)$  generates the label of  $k_i$  in layer  $\phi$ . A formal description of  $tag^\phi(k_i)$  is shown in Eq 2, in which  $C_\phi$  represents the the number of models in the  $\phi$ th layer.

$$tag^\phi(k_i) = j \quad j \in [1 : C_{\phi+1}] \quad (2)$$

$$\text{subject to: } \frac{T}{C_{\phi+1}} * (j) \leq label_i^m \leq \frac{T}{C_{\phi+1}} * (j + 1)$$

Second, we also describe what loss function is used in the training process. The loss value used in the training is defined as  $Loss = \sum_{\phi \in (1, \Phi+1)} \sum_{n \in (1, m_\phi)} (O_\phi^n - label_\phi^n)^2$ , which is the sum of the loss value of each model in the hierarchical architecture. We refer the output of  $n$ -th model in the  $\phi$ -layer to as  $O_\phi^n$ ,  $label_\phi^n$  to represent the corresponding labels, while  $m_\phi$  as the number of models in the  $\phi$ th layer.

### Algorithm 1 Labeling DLB training data

---

**Input:**  $K$  - key list of balls  
**Input:**  $T$  - number of positions on the hash circle  
**Input:**  $\Phi$  - number of layers in disperse stage  
**Input:**  $labels^i$  - labels for the models in the  $i$ th layer  
**Input:**  $indexof(k_i, K)$  - index of the element  $k_i$  in list  $K$   
**Input:**  $tag^\phi(k_i)$  - label of  $k_i$  for the model in  $\phi$ th layer.  
**Initialize:**  $Labels^i \leftarrow \{\}(i \in (1, \Phi + 1))$

- 1:  $K_s \leftarrow \text{Sort}(K)$
- 2: **for**  $k_i$  in  $K$  **do** //Create labels for models in mapping stage
- 3:    $label \leftarrow indexof(k_i, K) * (T/sizeof(K))$
- 4:    $labels^{\Phi+1} \leftarrow Labels^{\Phi+1} \cup label$
- 5: **end for**
- 6: **for**  $\phi$  in  $\Phi$  **do** //Create labels for models in disperse stage
- 7:   **for**  $k_i$  in  $K$  **do**
- 8:      $label \leftarrow tag^\phi(k_i)$
- 9:      $labels^\phi \leftarrow Labels^\phi \cup label$
- 10:   **end for**
- 11: **end for**
- 12: **return**  $\{labels^1, ..., labels^{\Phi+1}\}$

---

## IV. EVALUATION

In this section, we compare the effectiveness of load balancing between DLB and the following widely used and classical load balancing approaches:

- Consistent Hashing(CH). CH hashes the balls and bins into a unit circle, and uses the hash values to create a circular order of balls and bins. The placement decision are all based on the relative location among balls and bins.
- Consistent Hashing with Bounded Load(CHBL). CHBL is similar to CH, except that it uses a parameter to try to keep the balls uniformly distributed among different bins.

In our design each sub model of DLB has 3 fully connected layer, and each layer has 8, 32, 64 neuros respectively. We use Adam [29] with learning rate of 0.01 to train all the sub models. For CH and CHBL, we also combine them with different hash implementations, such as BKDR Hash [12], Python Hash [14], and Murmur Hash [13].



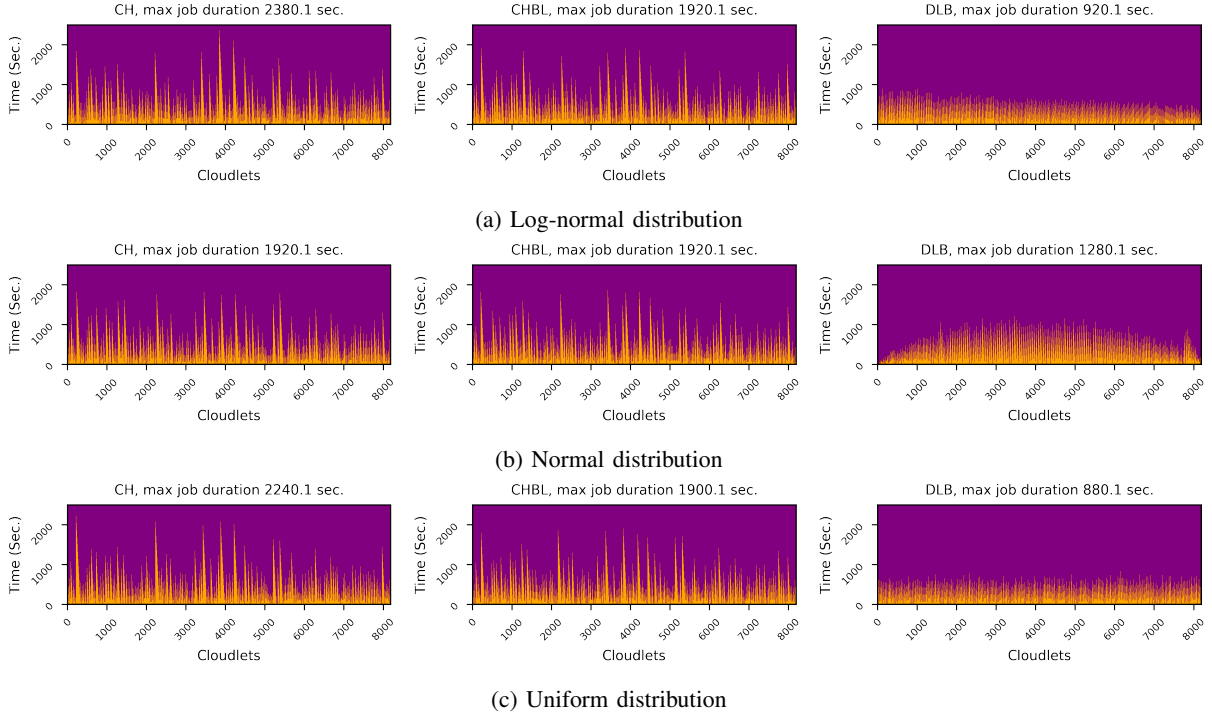


Fig. 3: Compare the effectiveness of different load balancing mechanisms in a practical Cloud environment created by CloudSim.

#### A. Setup

In this subsection, we describe the experimental setup, including the hardware and software environment, as well as the data sets and metrics used throughout the measurements.

1) *Environment*: The experiments are carried on a machine with 64GB main memory and one 2.6GHZ Intel(R) i7 processors. Each test is run 10 times and the median of the results are shown in this section.

2) *Data sets*: In order to measure how different distributions of the input data set can affect the effectiveness of DLB, the synthetic data sets used in Section IV-B are generated under three most commonly observed distributions: uniform distribution, normal distribution, and log-normal distribution. Each distribution has two data sets, one for testing and the other for training. Each data set consists of 20,000,000 balls while each ball is a double-precision digit which represents a key of client workload in load balancing scenario. A 4TB data set collected from a radio monitoring center is also used in our experiments. This data set consists of 100,000 records, and each record has 13 features.

3) *Measurement metrics*: The effectiveness of a load balancing mechanism is measured by the standard deviation among the load of different bins on the hash circle (std). Therefore, the smaller the std value is, the more effective the load balancing mechanism is. Formally, the *std* can be calculated as  $std = \sqrt{\frac{\sum_{j=1}^n (load_j - \frac{m}{n})^2}{n}}$ , in which,  $load_j$  refers to the number of balls assigned to bin  $j$ , while  $m$  and  $n$  are the total number of balls and bins respectively.

#### B. Analysis

1) *CloudSim based evaluation*: In this set of experiments, we deploy DLB, CH, and CHBL on a practical Cloud environment created by CloudSim [22], and compare their effectiveness to balance the workloads among a number of servers. In the simulated Cloud environment, 64 machines are hosted within a single data center. Each machine has 4 threads, 1 GB memory, 10 GB of storage, and 1Gbps network bandwidth. On the client side, 8192 workloads (cloudlets) with different distributions are created. The distribution is based on the ID of each workload, which is also the key used for mapping. Each server can run at most 4 workloads simultaneously. When wait list of a server is full and an additional workload is assigned, this workload needs to be loaded by the next spare server in the hash circle. All the workloads are submitted to the data center in one batch, and each workload takes 20 seconds CPU time to finish.

Fig 3 shows the actual finishing time of each workload when different load balancing mechanisms are used. The x-axis represents the ID of each workload while the y-axis shows the actual finishing time. It can be observed that, no matter what the workload distribution is, the heights of lines in figures corresponding to DLB is much lower and smoother. This means that, when DLB is used, workloads can finish in a shorter and more balanced time. While in CH and CHBL cases, some workloads takes much longer time to finish than the other ones. This is due to the imbalanced assignment of workloads, which causes some servers to become the hotspots. Therefore, it takes much longer for workloads on these servers

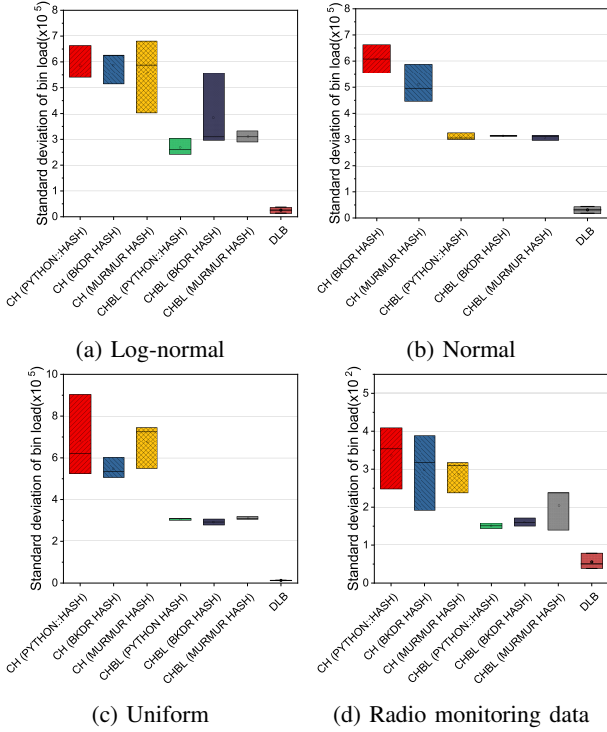


Fig. 4: Standard deviation of bin load using different load balancing mechanisms and distributions of input data sets

to finish. Considering the exact running time of the longest job (i.e., max job duration) in each scenario, DLB is able to reduce such time by 61.3% over CH and 52% over CHBL under log-normal distribution, 33% over both CH and CHBL under normal distribution, 60.7% and 53.68% when compared with CH and CHBL respectively under uniform distribution.

2) *Load Balancing*: Figure 4 compares the *std* of DLB with CH and CHBL based methods. The *std* is used to reflect how much an actual distribution of the balls on the hash circle deviates from an idea uniform distribution. For each experiment, we collect the average result as well as the distribution of 10 runs and plot them in Figure 4. It can be observed that compared with other methods, DLB has the lowest value of *std* regardless of the data distributions. For example, when the real-world data set is used, average *std* value of DLB for the 10 runs is 78, while that of other methods, such as CH(with Python Hash) and CH(with BKDR Hash) are 337 and 299 respectively, which are 3.32x and 2.83x larger than that of DLB.

## V. CONCLUSIONS

Existing hash function based load balancing mechanisms cannot perform well when the input data is skewed. In this paper, we proposed DLB, a Deep Learning based load Balancing mechanism, to address this problem. DLB replaces the hash functions in traditional load balancing mechanisms with deep learning models. Given the constant time of model inferencing, using a learned model does not introduce additional runtime overhead compared with using a hash

function. We implemented DLB and deployed it on a practical Cloud environment using CloudSim. Experimental results show that, compared to traditional hash function based load balancing mechanisms, DLB is able to achieve more balanced and stable results even when the input data is skewed.

## REFERENCES

- [1] EC Amazon. Amazon web services. Available in: <http://aws.amazon.com/es/ec2/>(November 2012), page 39, 2015.
- [2] Ibm cloud. <https://www.ibm.com/cloud>,
- [3] Google cloud. <https://cloud.google.com/>.
- [4] Microsoft azure. <https://azure.microsoft.com/en-us/>.
- [5] Bin Jiang and Tao Jia. Exploring human mobility patterns based on location information of us flights. *arXiv preprint arXiv:1104.4578*, 2011.
- [6] Filippo Radicchi. Human activity in the web. *Physical Review E*, 80(2):026118, 2009.
- [7] Elaheh Gavagsaz, Ali Rezaee, and Hamid Haj Seyyed Javadi. Load balancing in join algorithms for skewed data in mapreduce systems. *The Journal of Supercomputing*, 75(1):228–254, 2019.
- [8] Joanna Berlinska and Maciej Drozdowski. Comparing load-balancing algorithms for mapreduce under zipfian data skews. *Parallel Computing*, 72:14–28, 2018.
- [9] ThomasH.Cormen. . . [etal. *Introduction to algorithms*. 2002.
- [10] Shoichi Hirose and et al.
- [11] Michael Coles and Rodney Landrum. *Asymmetric Encryption*. 2009.
- [12] Arash Partow. Bkdr hash in "the general hash functions library", 2020.
- [13] Austin Appleby. Murmurhash3 on github", 2020.
- [14] Google Inc. The python standard library, 2020.
- [15] Dirk Brockmann, Lars Hufnagel, and Theo Geisel. The scaling laws of human travel. *Nature*, 439(7075):462, 2006.
- [16] Tim Kraska et al. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, 2018.
- [17] Wenkun Xiang and et al. Pavo: A rnn-based learned inverted index, supervised or unsupervised? *IEEE Access*, 7:293–303, 2019.
- [18] Tim Kraska and et al. Sagedb: A learned database system. In *CIDR 2019, 9th Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*, 2019.
- [19] Alex Galakatos and et al. Fiting-tree: A data-aware index structure. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 1189–1206, 2019.
- [20] David R. Karger and et al. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 654–663, 1997.
- [21] Vahab S. Mirrokni and et al. Consistent hashing with bounded loads. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 587–604, 2018.
- [22] Rodrigo N Calheiros and et al. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1), 2011.
- [23] David R. Karger and Matthias Ruhl. Simple efficient load-balancing algorithms for peer-to-peer systems. *Theory Comput. Syst.*, 39(6).
- [24] David Thaler and Chinya V. Ravishankar. Using name-based mappings to increase hit rates. *IEEE/ACM Trans. Netw.*, 6(1):1–14, 1998.
- [25] John Lamping and Eric Veach. A fast, minimal memory, consistent hash algorithm. *CoRR*, abs/1406.2294, 2014.
- [26] Roberto Grossi and Luca Versari. Round-hashing for data storage: Distributed servers and external-memory tables. In *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, pages 43:1–43:14, 2018.
- [27] Wei Wang and Chinya V. Ravishankar. Hash-based virtual hierarchies for scalable location service in mobile ad-hoc networks. *MONET*, 14(5):625–637, 2009.
- [28] Xiaoke Zhu, Taining Cheng, Qi Zhang, Ling Liu, Jing He, Shaowen Yao, and Wei Zhou. Nn-sort: Neural network based data distribution-aware sorting. *arXiv preprint arXiv:1907.08817*, 2019.
- [29] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.