Demonstrating Waffle: A Self-driving Grid Index

Dalsu Choi
Korea University
Seoul, Korea
dalsuchoi@korea.ac.kr

Hyunsik Yoon
Korea University
Seoul, Korea
hyunsikyoon@korea.ac.kr

Hyubjin Lee
Korea University
Seoul, Korea
hyubjinlee@korea.ac.kr

Yon Dohn Chung*
Korea University
Seoul, Korea
ydchung@korea.ac.kr

ABSTRACT

This paper demonstrates Waffle, a self-driving grid indexing system for moving objects. We introduce system architecture, system workflow, and user scenarios. Waffle enables the management of moving objects with less human effort while automatically improving performance.

PVLDB Reference Format:
Dalsu Choi, Hyunsik Yoon, Hyubjin Lee, and Yon Dohn Chung. Demonstrating Waffle: A Self-driving Grid Index. PVLDB, 16(12): 3954 - 3957, 2023.
doi:10.14778/3611540.3611594

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://dalsuchoi.github.io/Waffle.

1 INTRODUCTION

Moving object-based services are getting important and popular. Ride-sharing services, navigation services, autonomous driving, delivery services, and micro-mobility services are based on moving objects. The main feature of moving objects is that object positions, defined by latitude and longitude, may keep changing. When managing moving objects, the feature results in the two following challenges. First, location-updates should be handled efficiently while also considering scan queries. Second, various object distributions should also be considered.

To overcome these challenges, Waffle, a self-driving grid indexing system, has been proposed [4]. The main insights of Waffle are (1) to keep a regular grid to handle location-updates efficiently and (2) to change a grid definition according to object distribution. Specifically, Waffle handles the following problems: how to define an index structure, when to change a grid definition, and how to rebuild an index without blocking user queries.

Waffle consists of three main parts: a Waffle index, WaffleMaker, and a regrid. A **Waffle index** is an in-memory grid index optimized for location-updates [4]. A Waffle index divides a geographical space into fixed-sized cells, and neighboring cells are grouped into a chunk to improve cache efficiency. Waffle supports four query types for a Waffle index: insertion, deletion, range, and k-NN queries. A Waffle index is defined by five configuration knobs, which have a huge impact on the performance of Waffle. However, it is not trivial to determine a decent knob setting considering a lot of factors including object distribution. Therefore, Waffle includes an automatic configuration tuner, **WaffleMaker** [4]. For the current object distribution, WaffleMaker determines a knob setting and whether to rebuild a Waffle index. In case of rebuilding a Waffle index with the new knob setting, Waffle does not block user queries during the rebuilding based on a concurrency control scheme, and the mechanism is called a **regrid** [4].

We introduce the implementation details of Waffle, a mechanism to tune hyperparameters of Waffle, and user experiences when managing moving objects, not covered in the paper [4]. Section 2 introduces the system architecture and workflow of Waffle. Section 3 demonstrates Waffle, and related studies are introduced in Section 4. Section 5 concludes the paper.

2 SYSTEM ARCHITECTURE

We introduce the architecture of Waffle. Waffle consists of six components: a query parser, a transaction manager, a Waffle index manager, a lock manager, a regrid manager, and WaffleMaker, as shown in Figure 1.

Given an insertion/deletion/range/k-NN query, a **query parser** parses the query, extracts necessary information including a query type and query parameters, and requests a transaction manager to process the query.

A **transaction manager** constructs a transaction with query information from a query parser. For the same query, a transaction manager may construct different transactions depending on whether Waffle is performing a regrid. Specifically, during a regrid, a transaction manager considers not only an original index but also a new index, if required.

A **Waffle index manager** is responsible for a single Waffle index. A Waffle index manager processes operations of a transaction for the manager's Waffle index. The manager also keeps statistics to calculate a reward, including query processing times, the number of processed queries, and memory usage. A Waffle index manager includes a **lock manager**. A lock manager grants and releases a lock considering lock compatibility. Because each Waffle index may have a different chunk definition, and a set of chunks with the same chunk coordinate is the unit of a lock, each Waffle index manager has its own lock manager. During a regrid, two Waffle index managers exist.

Waffle launches a **regrid manager** to perform a regrid. The regrid manager receives a new knob setting from WaffleMaker and generates a new Waffle index manager. For each object in the original index, the regrid manager processes a transfer transaction, which deletes the object from the original index and inserts the object into the new index. After processing all the transfer transactions, the regrid manager replaces the original Waffle index manager with the new Waffle index manager, and Waffle terminates

*Corresponding author.

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611594

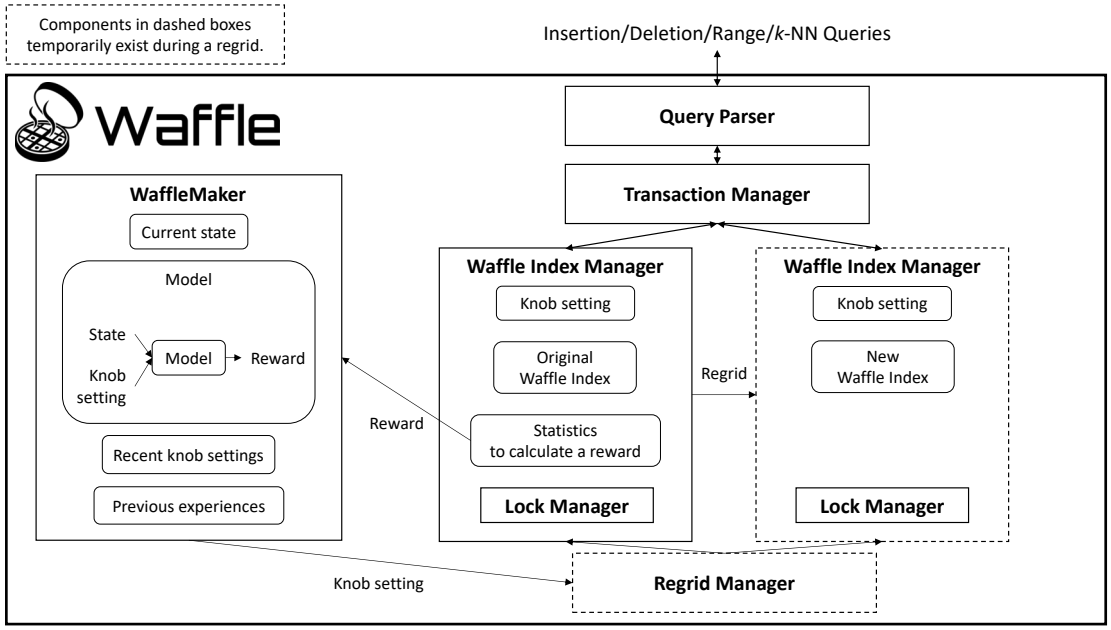
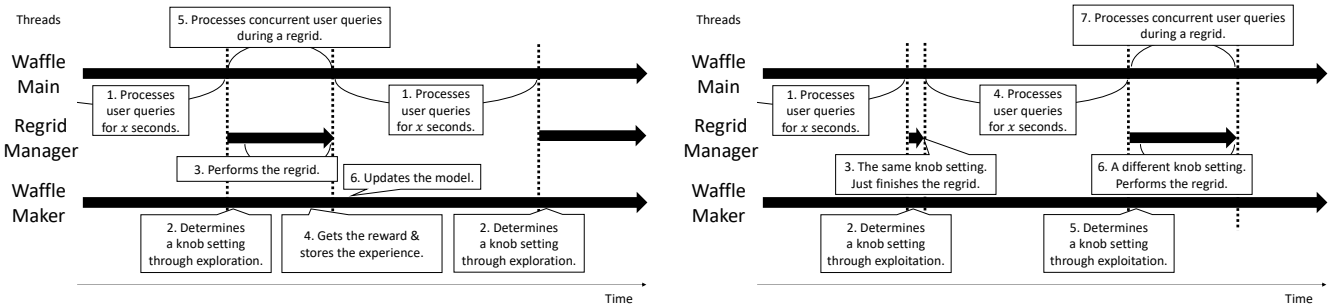


Figure 1: Architecture of Waffle



(a) Before a WaffleMaker model converges **(b) After a WaffleMaker model converges**

Figure 2: Workflow of Waffle

the regrid manager. A regrid manager temporarily exists during a regrid and runs in a separate thread.

WaffleMaker is an automatic configuration tuner that determines a new knob setting for a regrid. A WaffleMaker state summarizes the current objects in a geographical space and is defined as a fixed-sized grid, which is not related to a Waffle index. Each state cell maintains the number of corresponding objects. A WaffleMaker model is a convolutional neural network; inputs are a state and a knob setting, and an output is an expected reward, which represents the performance of Waffle including query processing times and memory usage. WaffleMaker determines a knob setting using exploration or exploitation [4]. A regrid manager requests a knob setting to WaffleMaker, and WaffleMaker determines a knob setting for the current state. Afterward, WaffleMaker obtains a reward for the knob setting. WaffleMaker stores the state, the knob setting, and the reward as a new experience and utilizes the previous experiences to update the model. Specifically, WaffleMaker samples $|batch|$ previous experiences using a prioritized experience replay [9] and updates the model with a given learning rate.

Waffle was implemented using C++14 and PyTorch 1.13.1 with the PyTorch C++ frontend [1].

2.1 Workflow

We introduce the workflow of Waffle in Figure 2. Waffle executes maximally three threads at the same time: a Waffle main thread, a thread for a regrid manager, and a thread for WaffleMaker. The Waffle main thread runs a query parser, a transaction manager, and maximally two Waffle index managers [2]. The workflow depends on convergence of a WaffleMaker model.

We first introduce the workflow before a WaffleMaker model converges. Waffle launches a new regrid manager x seconds after the previous regrid. The previous work [4] utilizes the number of processed user queries as a condition to start a new regrid, which is not intuitive for a user. Instead, the regrid condition is set based on time, which is more user-friendly. WaffleMaker determines a new

[1] https://pytorch.org/

[2] Waffle processes user queries in a single thread in the current implementation but can be extended to process user queries in multiple threads.

knob setting through exploration [4] and passes the knob setting to the regrid manager. While the regrid manager performs the regrid, WaffleMaker obtains the reward for the previous knob setting from the user queries processed for the x seconds and stores the new experience. Waffle processes concurrent user queries during the regrid based on a concurrency control scheme. WaffleMaker updates the model using the previous experiences. After finishing the regrid, Waffle processes user queries for the next x seconds and repeats the process until the WaffleMaker model converges.

The workflow after a WaffleMaker model converges is as follows. Waffle starts a new regrid manager x seconds after the previous regrid. WaffleMaker determines a candidate knob setting through exploitation [4], which selects one of the knob settings determined by the recent explorations. If the candidate knob setting is the same as the current knob setting, Waffle does not have to perform the regrid. The knob setting is still expected to perform the best among the recent knob settings. After processing user queries for the next x seconds, Waffle launches a new regrid manager, and WaffleMaker selects a candidate knob setting with the highest expected reward among the recent knob settings. If the candidate knob setting is different from the current knob setting, Waffle performs the regrid. Waffle naturally determines when to perform a regrid based on the exploitation method.

3 DEMONSTRATION

Waffle aims to minimize human efforts when managing moving objects. Waffle does not require administrators to consider a lot of factors including object distribution. The interface for Waffle was implemented using React [3], Chart.js [4], and Restbed [5].

3.1 Dataset

We first introduce the dataset used in the demonstration. We generated a synthetic dataset in a similar way to the work [4]. First, we extracted road network data [5] in Los Angeles, where the latitude values were $[33.8449489, 34.3242765]$, and the longitude values were $[-118.7660027, -117.7932931]$. Moving objects were generated on the roads based on the concept of an **episode**. An episode was defined as follows. $|base|$ objects were placed on random roads or at the last positions of the previous episode and randomly moved along roads. We gradually inserted $|random|$ objects on random roads and $|center|$ objects around the center of the geographical space, while the objects already in the space kept moving. After inserting $|random|$ and $|center|$ objects, we randomly moved $|base|$, $|random|$, and $|center|$ objects. Then, we gradually deleted $|random|$ and $|center|$ objects from the space, and one episode was completed. We used $|base| = 100,000$, $|random| = 900,000$, and $|center| = 1,000,000$. In the middle of insertion and deletion queries, we inserted a range or k-NN query in the same manner as that of the work [4]. The example object distributions are shown in Figure 3 [6].

[3] https://reactjs.org/
[4] https://www.chartjs.org/
[5] https://github.com/Corvusoft/restbed
[6] In the interface, we excluded visualization of object distribution because of the overhead from visualizing millions of objects, which is a future work.

(a) $|base|$ **(b)** $|base| + |random| + |center|$

Figure 3: Object distribution in LA dataset

We generated multiple episodes of queries and stored them on storage. Waffle read and processed the queries one by one. Waffle can be extended to receive queries through network.

3.2 User Scenarios

A user manages moving objects from Uber, Lyft, Lime, Bird, Google Maps, Waze, Grubhub, DoorDash, Uber Eats, Postmates, and Tesla.

Step (1) User Preference. A user provides preference: w_{time}, w_{memory}, and x. w_{time} and w_{memory} determine a trade-off between query processing times and memory usage, where $w_{time} + w_{memory} = 1$. If w_{time} is set higher than w_{memory}, Waffle tries to reduce query latency while giving less consideration to memory usage. A user also provides x as a condition to start a new regrid.

Step (2) Tuning Hyperparameters. Waffle provides a mechanism to tune hyperparameters including a learning rate and $|batch|$. The main purpose of the mechanism is to reduce user efforts and to tune hyperparameters while processing user queries. The mechanism to tune hyperparameters is as follows.

Step (2-1). Waffle suggests hyperparameter values when a user clicks 'Try' button. For each hyperparameter, Waffle randomly samples a value from the predefined range [3] [7]. For the first hyperparameter values only, Waffle tries random knob settings without considering the exploration method until collecting base experiences. Waffle follows the workflow in Figure 2(a) with the selected hyperparameter values.

A user can check the current knob setting from the interface. The interface shows the average values of query processing times, memory usage, rewards, and losses during an episode to help a user to monitor the performance of Waffle. A user can recognize that the performance of Waffle is getting better as time goes on because of appropriate knob settings. The number of objects in the space, that of performed regrids, and that of processed user queries are also shown in the interface to share the progress with a user.

Step (2-2). A user evaluates the hyperparameter values by clicking 'Evaluate' button when the WaffleMaker model converges. Based on the graphs, especially for rewards and losses, a user can easily determine convergence when the performance of Waffle no longer improves. Waffle follows the workflow in Figure 2(b) and observes the performance of Waffle. Specifically, Waffle keeps recording rewards during the evaluation as performance measures. After evaluating the hyperparameter values, a user determines

[7] Different techniques can be applied instead of random search, and this is out of the main focus of this paper.

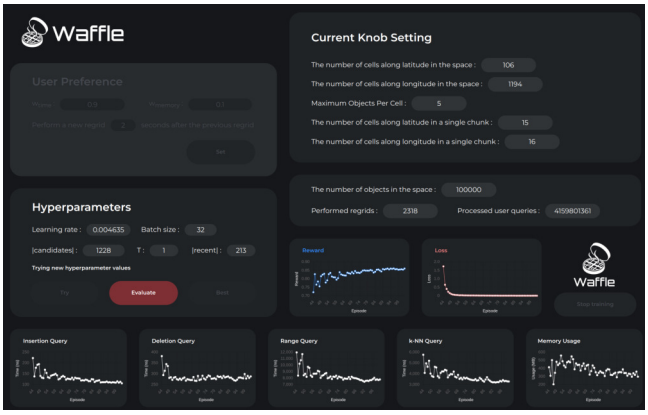


Figure 4: The Waffle interface

whether to try new hyperparameter values (Step (2-1)) or to select the best hyperparameter values among the previous trials (Step (3)), and Waffle records the average reward during the evaluation with the evaluated hyperparameter values.

Before proceeding to Step (2-1) or Step (3), Waffle initializes the WaffleMaker model. Experiences from previous hyperparameter values, except for the base experiences, are not reused for the next step to improve fairness and to remove dependency on previous hyperparameter values.

Discussion. The mechanism to tune hyperparameters is performed while processing user queries. In other words, Waffle does not tune hyperparameters using fixed training / validation / test dataset, and the tuning mechanism could be unfair for some selected hyperparameter values. It is an option to tune hyperparameters for pre-prepared dataset before starting Waffle. However, Waffle focuses on minimizing user efforts and increasing usability, and the tuning process is performed based on actual user queries.

Step (3) Running Waffle with the Best Hyperparameter Values. Waffle selects the best-performed hyperparameter among the trials at Step (2) by clicking 'Best' button. Waffle first follows the workflow in Figure 2(a). After the WaffleMaker model converges, a user can stop training the model by clicking 'Stop training' button, and then Waffle follows the workflow in Figure 2(b). Waffle keeps reporting the performance measures excluding a loss, and a user can restart training the WaffleMaker model if the performance is not satisfying by clicking 'Restart training' button, which was originally 'Stop training' button.

4 RELATED WORK

Learned spatial indexes based on a recursive model index [6] have been researched [8, 10]. Waffle defines index optimization as a configuration tuning problem while considering location-updates as the primary goal. Automatic database configuration has been studied [1, 2, 7, 11, 12]. WaffleMaker determines when to change a knob setting as well as a knob setting considering object distribution.

5 CONCLUSION

We demonstrated Waffle, a self-driving grid indexing system for moving objects. We introduced the system internals, workflow, and user scenarios. An attendee will be able to experience a novel approach to managing moving objects with less human effort. An attendee can intuitively interact with Waffle by inputting user preference, tuning hyperparameters, observing performance measures, and determining convergence of a WaffleMaker model.

For update-intensive workloads, especially from moving objects, Waffle proposes a novel approach to optimizing indexes by defining the optimization as a configuration tuning problem. This approach may be applied to different kinds of indexes, which require knob settings, in case of handling a lot of updates. To apply the approach, WaffleMaker may have different types of states and rewards. The mechanism for redefining an index, called a regrid, may be modified according to an index structure.

In addition, the paradigm of WaffleMaker, which maps continuous knob space to discrete knob space, may be utilized for a database configuration tuning problem. If a single knob setting is inappropriate because of changing data or workloads, the paradigm can naturally determine when to change a knob setting.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their time and the valuable comments. This work was supported by (1) the National Research Foundation of Korea (NRF) grant funded by the Ministry of Science and ICT (MSIT) (NRF-2020R1A2C2013286), (2) MSIT under the ICT Creative Consilience program (IITP-2023-2020-0-01819) supervised by the IITP (Institute for Information communications Technology Planning Evaluation), and (3) Basic Science Research Program through NRF funded by the Ministry of Education (NRF-2021R1A6A1A13044830).

REFERENCES

[1] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. 2017. Automatic Database Management System Tuning Through Large-scale Machine Learning. In *SIGMOD 2017*. ACM, 1009–1024.
[2] Dana Van Aken, Dongsheng Yang, Sebastien Brillard, Ari Fiorino, Bohan Zhang, Christian Billian, and Andrew Pavlo. 2021. An Inquiry into Machine Learning-based Automatic Configuration Tuning Services on Real-World Database Management Systems. *PVLDB* 14, 7 (2021), 1241–1253.
[3] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research* 13 (2012), 281–305.
[4] Dalsu Choi, Hyunsik Yoon, Hyubjin Lee, and Yon Dohn Chung. 2022. Waffle: In-memory Grid Index for Moving Objects with Reinforcement Learning-based Configuration Tuning System. *PVLDB* 15, 11 (2022), 2375–2388.
[5] Ahmed Eldawy and Mohamed F. Mokbel. 2019. Roads and streets around the world each represented as individual line segments. Retrieved from UCR-STAR https://star.cs.ucr.edu/?OSM2015/road-network, last access: 06/24/2023.
[6] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In *SIGMOD 2018*. ACM, 489–504.
[7] Guoliang Li, Xuanhe Zhou, Shifu Li, and Bo Gao. 2019. QTune: A Query-Aware Database Tuning System with Deep Reinforcement Learning. *PVLDB* 12, 12 (2019), 2118–2130.
[8] Jianzhong Qi, Guanli Liu, Christian S. Jensen, and Lars Kulik. 2020. Effectively Learning Spatial Indices. *PVLDB* 13, 11 (2020), 2341–2354.
[9] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2016. Prioritized Experience Replay. In *ICLR 2016*.
[10] Haixin Wang, Xiaoyi Fu, Jianliang Xu, and Hua Lu. 2019. Learned Index for Spatial Queries. In *MDM 2019*. IEEE, 569–574.
[11] Bohan Zhang, Dana Van Aken, Justin Wang, Tao Dai, Shuli Jiang, Jacky Lao, Siyuan Sheng, Andrew Pavlo, and Geoffrey J. Gordon. 2018. A Demonstration of the OtterTune Automatic Database Management System Tuning Service. *PVLDB* 11, 12 (2018), 1910–1913.
[12] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, Minwei Ran, and Zekang Li. 2019. An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning. In *SIGMOD 2019*. ACM, 415–432.