



# Catcher: A Cache Analysis System for Top-*k* Pub/Sub Service

Baolong Mei  
Zhengzhou University  
Zhengzhou, China  
blmeizzu@gs.zzu.edu.cn

Yafei Li  
Zhengzhou University  
Zhengzhou, China  
ieyfli@zzu.edu.cn

Wei Chen  
Zhengzhou University  
Zhengzhou, China  
chenweizzu@gs.zzu.edu.cn

Linshen Luan  
Zhengzhou University  
Zhengzhou, China  
llszzu@outlook.com

Guanglei Zhu  
Zhengzhou University  
Zhengzhou, China  
glzhu@gs.zzu.edu.cn

Yuanyuan Jin  
Zhengzhou University  
Zhengzhou, China  
jinyuanyuan@zzu.edu.cn

Jianliang Xu  
Hong Kong Baptist  
University  
Hong Kong SAR, China  
xujl@comp.hkbu.edu.hk

## ABSTRACT

Top-*k* Publish/Subscribe (TkPS) service is widely studied in spatial database, with various cache-based methods proposed to address its efficiency challenge in top-*k* result maintenance. These methods require in-depth exploration of relationships between cache updates and different factors (e.g., data distribution) to optimize cache performance. However, there is currently no system available that assists developers in conducting comprehensive cache analyses within TkPS services. We therefore introduce *Catcher*, a multi-functional cache analysis system designed for TkPS services. It not only enables users to intuitively analyze the entire maintenance process of top-*k* results but also aids in identifying bottlenecks and potential optimization spaces of caches. Catcher provides two user-friendly interfaces that allow users to employ simple and easy-to-use consoles to perform statistical analysis. Furthermore, Catcher offers the real-time evaluation of cache-based methods, providing users with instant analysis. We have demonstrated the usability of Catcher on real-world datasets. A short video of our demonstration can be found at <https://youtu.be/qI81HoypB0w>.

### PVLDB Reference Format:

Baolong Mei, Yafei Li, Wei Chen, Linshen Luan, Guanglei Zhu, Yuanyuan Jin, and Jianliang Xu. Catcher: A Cache Analysis System for Top-*k* Pub/Sub Service. PVLDB, 17(12): 4389 - 4392, 2024. doi:10.14778/3685800.3685882

### PVLDB Artifact Availability:

The source code, data, and/or other artifacts have been made available at <https://github.com/Keleton-Y/Catcher>.

## 1 INTRODUCTION

Publish/Subscribe (Pub/Sub) is a widely used messaging service where the platform matches subscriptions (data receivers) with incoming messages (data publishers) based on their criteria, and sends matched messages to subscribers. In recent years, Top-*k* Pub/Sub (TkPS) service has emerged as a variant of Pub/Sub, which

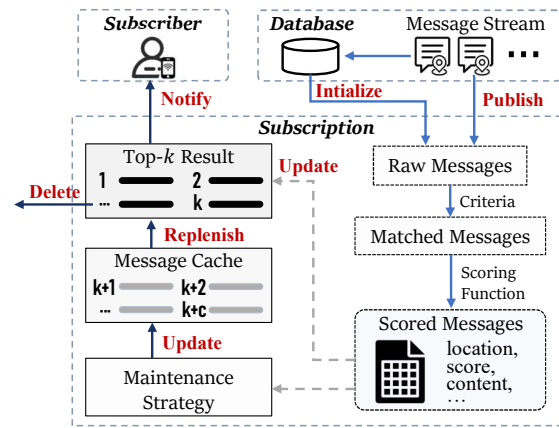


Figure 1: An illustration of the top-*k* result maintenance process using message cache in TkPS services.

returns up-to-date top-*k* ranked messages to subscribers instead of simply notifying all matched messages. It has widespread applications in spatial object recommendation. For example, TkPS services can identify the most relevant *k* shops for moving subscribers [8], deliver instant updates on the top-*k* trending terms within user-defined regions [1], and recommend top-*k* real-time tasks [5, 6] or top-*k* worker teams [4] in spatial crowdsourcing. The primary technical challenge of the TkPS service is to efficiently maintain massive subscriptions over a large-scale message stream. To overcome the challenge, a series of existing works [2, 3, 6, 7, 9–12] propose message cache (a.k.a. buffer or top-*k* refiller) models to accelerate the maintenance of subscriptions. As illustrated in Figure 1, the cache stores several candidate messages following the update of the top-*k* result, and its core component named maintenance strategy determines which messages to be retained. When messages in the top-*k* result become unmatched (i.e., no longer meet the criteria of the subscription), they are deleted and the cache helps replenish the result with stored messages, thus eliminating the need to frequently traverse the database to re-initialize the top-*k* result. Existing cache models in the TkPS service can be classified into three types: *i*) threshold-based method [11], which only stores messages with scores higher than a given threshold value; *ii*) capacity-based method [12], which maintains the top-*k<sub>m</sub>* results where *k<sub>m</sub>* > *k* is the cache capacity; *iii*) region-based method [3], which computes

\*Corresponding author: Yafei Li

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing [info@vldb.org](mailto:info@vldb.org). Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097. doi:10.14778/3685800.3685882

message score bounds and stores a list of partitioned areas that cover message sets.

Although caches are beneficial for TkPS services, designing and optimizing their maintenance strategies is quite challenging. Firstly, an effective strategy requires adapting cached messages in response to changes in factors like the quantity and geographical distribution of arrived messages. However, a reliable system is currently lacking to assist developers in exploring and establishing the connections between effective strategic decisions and these factors. Secondly, it is hard to analyze the bottleneck of the maintenance strategy by raw log files, because developers cannot intuitively identify when improper decisions occur and what factors lead to the unreasonable storage of candidates. For example, we cannot determine which messages are invalidly stored (i.e., never included in the top- $k$  result but included in the cache) until they are removed from the cache. This necessitates developers to conduct a retrospective analysis of the maintenance process. However, much of the intermediate data conducive to analysis has already been changed and not stored.

Motivated by the above discussion, we demonstrate *Catcher*, an advanced cache analysis system for TkPS service developers and managers. *Catcher* aims to empower users to effortlessly analyze the process of result maintenance, discover the deficiencies in maintenance strategies, and optimize their cache models. By inputting message streams and subscription sets, *Catcher* can replicate the computational process of TkPS services. Throughout this process, it catches a wealth of operational details. *Catcher* then analyzes the data, provides suggestions, and visualizes the analysis results to users through two user-friendly interfaces. Moreover, users can evaluate and compare the performance of multiple cache models simultaneously on uploaded datasets, and access instant analysis. Three types of discussed cache models and a novel learning-based method are deployed in *Catcher* to meet diverse user needs.

## 2 SYSTEM OVERVIEW

In this section, we present several preliminaries about the Top- $k$  Pub/Sub system, outline the overall architecture of *Catcher*, and discuss some cache models deployed in *Catcher*.

### 2.1 Preliminaries

*Catcher* is designed to conduct an in-depth analysis of cache models by simulating TkPS services. We provide the following definitions and concepts related to TkPS services.

**Message.** A message  $m \in M$  is denoted by a tuple  $m = (l, t, c)$ , where  $m.l$  is a location,  $m.t$  is the arrival time, and  $m.c$  is the message content. In subscriptions, messages are ranked based on their location and content with respect to a given scoring function. The message content  $m.c$  varies across different datasets, which can include a set of keywords [1], a destination with deadline [5, 6], and other objects [9]. We store the content data in JSON format to adapt to its heterogeneous nature.

**Subscription.** A subscription is denoted by a tuple  $s = (l, k, f, R)$ , where  $s.l$  is a location,  $s.k$  is the number of messages to be returned as the result,  $s.f : M \mapsto \{0, 1\}$  is a Boolean function that determines whether messages meet the criteria of the subscription, and  $s.R : M \mapsto \mathbb{R}$  is a parameterized scoring function that scores the matched

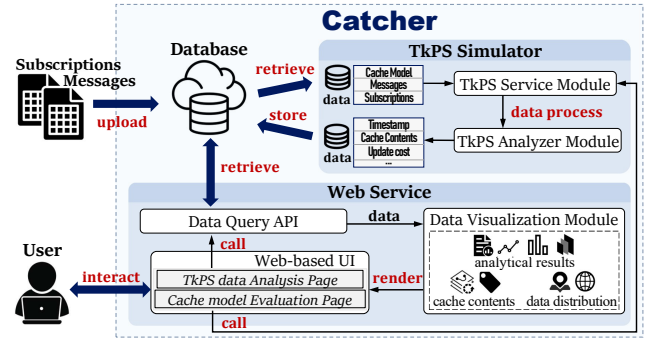


Figure 2: Overview of the Catcher architecture.

messages. It is important to note that the parameters in  $s.f$  and  $s.R$  differ among various subscriptions.

**TkPS service.** Considering a subscription set  $S$  and an incoming message stream  $M$ , the TkPS service maintains a latest list  $\mathcal{R}_s$  of size  $s.k$  for each subscription  $s \in S$ , where  $\mathcal{R}_s \subseteq M$ ,  $|\mathcal{R}_s| = s.k$ , and  $\forall m \in \mathcal{R}_s, m' \in M \setminus \mathcal{R}_s, s.R(m) \geq s.R(m')$ .

To efficiently cope with high data streaming rates, it is essential to minimize the computational cost of supporting TkPS service. Here, we introduce the message cache as a solution.

**Message cache.** A message cache associated with the subscription  $s$  is a data structure that maintains a set of messages  $C_s \subseteq M \setminus \mathcal{R}_s$ . When  $|\mathcal{R}_s| < s.k$ ,  $C_s$  is guaranteed to hold a candidate message  $m$  satisfying  $\forall m' \in M \setminus (\mathcal{R}_s \cup \{m\}), s.R(m) \geq s.R(m')$  to replenish it. If  $|\mathcal{R}_s| < s.k$  and  $C_s = \emptyset$ , the top- $k$  result and the cache should be both re-initialized to ensure the subsequent result maintenance.

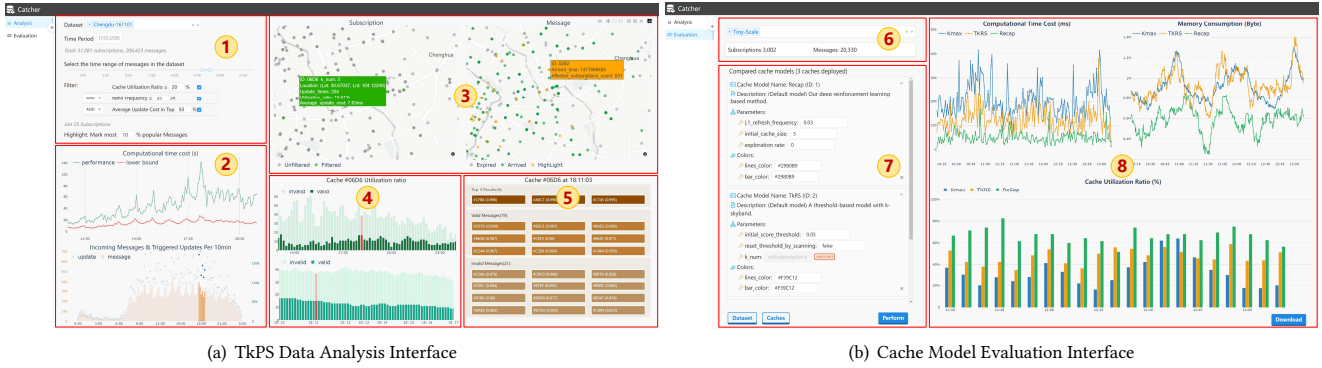
The performance of the cache depends on its utilization, which refers to the proportion of stored messages that are ultimately added to the top- $k$  results. To accurately measure this metric, *Catcher* records the sequences of tasks appearing in  $\mathcal{R}_s$  and  $C_s$ . After the TkPS service termination, it categorizes messages as *valid* or *invalid* based on their presence in  $\mathcal{R}_s$ . The proportion of valid messages in the cache  $C_s$  indicates the cache utilization ratio.

### 2.2 Architecture

Figure 2 depicts the overall architecture of *Catcher*. It consists of the following three key components.

**Database.** Tables in the *Catcher* database are divided into two parts. The first part is associated with uploaded message data and subscription data, they are initially pre-processed to add some metadata when uploaded, such as their temporal and spatial extents. Then, they serve as inputs for the TkPS simulator. The second part consists of derived data generated by the TkPS simulator, which is retrieved by the data query API.

**TkPS Simulator.** This is the core component of *Catcher*, which comprises the TkPS service module and the TkPS analyzer module. The TkPS service module is accessible to users via the web-based UI. It fetches subscription and message data from the database, loads the selected cache model, and then proceeds to simulate the entire computational process of the TkPS service in chronological order. At each step, it collects data such as timestamps, cached messages, subscription results, overall memory usage, and more. The collected



(a) TkPS Data Analysis Interface

(b) Cache Model Evaluation Interface

Figure 3: Demonstration of Catcher.

data is then cleansed and categorized. Subsequently, the analyzer module receives the data and generates additional derived data, including utilization ratios, total computational costs, and the number of update operations executed for each cache. Finally, this data is uploaded to the database for subsequent use by the web service.

**Web Service.** After the TkPS service is simulated, users can access pre-processed data and start their analysis through the web service. It comprises multiple visualization panels that concurrently present data distributions, cache statuses, subscription information, and more. Furthermore, the web offers some user-friendly interactive components to call data query API, allowing users unfamiliar with SQL-like languages to efficiently query TkPS data and analyze results without resorting to coding.

### 2.3 Cache Models

Catcher supports the analysis of three mainstream cache models, all of which have been deployed within the system. Here, we briefly discuss the technical details and the optimization potential of them.

Threshold-based methods construct a partial  $k$ -skyband [11] to manage candidate messages and adaptively set its threshold value as the score of the  $k$ -th message in  $\mathcal{R}_s$  when invoking the latest top- $k$  result re-initialization. Their limitation lies in focusing solely on message scores while ignoring the update patterns of  $\mathcal{R}_s$ , namely, how many messages have been recently added to or removed from  $\mathcal{R}_s$ . Assisted by Catcher, users can identify low-utilization caches caused by inappropriate threshold settings and examine the update patterns in corresponding  $\mathcal{R}_s$  to optimize their methods.

Capacity-based methods [12] develop a cost function for caches, which estimates the computational cost of cache maintenance given a capacity and the update patterns of  $\mathcal{R}_s$ . They determine cache capacities by minimizing this cost. However, these methods assume that subscriptions in close proximity exhibit similar update patterns in  $\mathcal{R}_s$ , which overlooks the influence of subscription parameters (e.g., criteria, custom scoring functions) on the distribution of matched messages. In Catcher, users can observe how these parameters affect the updates of  $\mathcal{R}_s$  and  $C_s$  by tracking cache status and retrieving subscription information, which potentially reveals issues in the capacity settings of existing methods.

Region-based methods [3] initially partition the road network into grids and then calculate the upper and lower bounds of message

scores within each grid. They maintain a grid list for each subscription, ensuring that the  $(k + 1)$ -th message of the subscription resides in one of these grids. By observing the cache performance under different grid size settings in Catcher, users can identify the suitable grid size for each region to achieve an optimal trade-off between memory and computational time costs.

Additionally, we propose a learning-based approach that may provide valuable insights for users. The idea behind this approach is to design a reinforcement learning agent for the cache, which captures factors (e.g.  $\mathcal{R}_s$  and  $C_s$  content, the update patterns of  $\mathcal{R}_s$ ) within the TkPS service as environmental states and trains through the Catcher simulator. The actions of the agent, based on different cache model types, can involve setting thresholds, capacities, or grid sizes. The computational cost of the cache at each time step determines its reward. We have deployed a method in Catcher, named Recap, which combines the capacity-based cache with a learning-based module. Recap has shown significant efficiency improvements within Catcher, demonstrating the potential of enhancing cache models through learning-based approaches.

## 3 DEMONSTRATION

We showcase Catcher by analyzing the performance of caches in a TkPS service introduced in our prior work [6]. This system is proposed to disseminate top- $k$  ride order messages to spatial crowd-sourcing workers. The message data, collected on 11/01/2016 by DiDi in Cheng Du, comprises 209,423 ride orders. The subscription dataset consists of over 30,000 distinct subscriptions.

### 3.1 User Interfaces

Users can engage with Catcher via two web pages: the TkPS data analysis interface and the cache model evaluation interface. Figure 3 shows the screenshots of them.

As shown in Figure 3(a), the TkPS data analysis interface consists of five panels. Panel ① serves as a control center to manage displayed datasets and execute data retrieval operations. Panel ② is a monitor that logs the computational time, incoming message count, and triggered update quantity within the simulated service. Catcher further estimates the potential minimum maintenance time consumption by eliminating the impact of invalid storage and then renders it in red lines. Moving to Panel ③, users can explore the

geographical distribution of subscriptions and messages across the road network. Hovering over data points reveals their detailed information. Panel ④ is two bar charts that illustrate the utilization dynamics of the cache at different granularities: snapshots taken every ten minutes and with every update. In these charts, the dark-colored segments represent valid messages, while the lighter ones denote invalid messages. Panel ⑤ is a table of cached messages associated with the snapshots in Panel ④.

As shown in Figure 3(b), the cache model evaluation interface comprises three different panels. Panel ⑥ aids users in selecting datasets as the input for the simulator module. Panel ⑦ is a cache manager that helps users add, remove, and modify cache models. Within the manager, adjustable parameters of cache models and their display colors in the charts can be easily modified. When users click the "Perform" button, the simulation begins. The ensuing real-time results are dynamically rendered in Panel ⑧. Users are provided with charts representing three key metrics: computational time cost, memory consumption, and cache utilization ratio. Following the termination of the process, users can download the results by the "Download" button for local review and analysis.

### 3.2 User Scenarios

We further demonstrate how TkPS service developers can analyze caches using Catcher through the following three scenarios.

**Scenario 1: Cache Performance Analysis.** In the TkPS data analysis interface, users can dynamically select the time period and dataset to be analyzed on Panel ①, and conduct cache performance analysis from three different perspectives: all caches, a group of caches, and individual caches. Specifically, users can access service statistics on Panel ②, which helps them analyze the overall performance of caches. Then, users can leverage data retrieval operations provided by Panel ① to filter caches on Panel ③, thereby obtaining the distribution of caches with different performances. For example, they can search for a group of caches with both low utilization and high re-initialization frequency. Hovering over them reveals specific information about the caches. These operations aid users in analyzing which subscriptions are hard to maintain under the current cache model. Finally, clicking on a cache will display its utilization changes throughout the entire maintenance process on Panel ④, revealing which time periods the cache performs better or worse, facilitating users in analyzing the relationship between time periods and cache performance.

**Scenario 2: Maintenance Strategy Analysis.** Catcher provides a cache tracking tool that consists of Panel ④ and Panel ⑤ to aid users in scrutinizing each strategic decision. The chart in Panel ④ (the lower side one) displays snapshots of caches after each update, where each snapshot related to a timestamp is represented as a bar. Hovering over these snapshots triggers Panel ⑤ to load the corresponding analysis results for that moment, including the cached messages and top-*k* results. The messages in the cache are further categorized into valid and invalid messages based on the methods discussed in Section 2.1. This information is particularly helpful for threshold-based and capacity-based strategies. Based on the lowest message score and the total number of messages in the cache, users can identify the specific thresholds and capacities set by the maintenance strategy for the cache. Additionally, based on the lowest

score and total number of valid messages in the cache, they can determine the optimal settings for thresholds and capacities at that moment. This aids them in conducting comprehensive analyses and optimizing maintenance strategies.

**Scenario 3: Cache Model Analysis.** Catcher also provides an interface for evaluating and analyzing cache models. To begin the analysis, users should select a dataset from Panel ⑥. Then, they can manage existing cache models through Panel ⑦ and select multiple methods to compare. We provide four default cache models in Catcher: PCL [3], TkRS [5], kmax [12], and a novel learning-based method Recap. Users can also upload custom maintenance strategies. Clicking the "Perform" button deploys selected models in the simulator and initiates the cache evaluation process, and the performance of different caches is displayed in real time on Panel ⑧. This allows users to intuitively analyze which cache models exhibit advantages under these datasets. Additionally, by uploading datasets derived from TkPS applications, users can also learn about the performance of different cache models in their TkPS services.

## 4 CONCLUSION

Motivated by the challenge of developing and optimizing cache models in TkPS services, we demonstrate the cache analysis system *Catcher*. It helps users improve their techniques by analyzing the performance of cache models and understanding the impact of different factors. We demonstrate three scenarios using a real-world dataset to illustrate the usability of Catcher.

## ACKNOWLEDGMENTS

This work is supported by NSFC Grants 62372416, 61972362, and 62302460, HNSF Grant 242300421215, CPSF Grant 2022TQ0297, Hong Kong RGC Grant (R1015-23), and Guangdong Basic and Applied Basic Research Foundation (2023B1515130002).

## REFERENCES

- [1] Lisi Chen, Shuo Shang, Christian S Jensen, Jianliang Xu, Panos Kalnis, Bin Yao, and Ling Shao. 2020. Top-k term publish/subscribe for geo-textual data streams. *VLDBJ* 29 (2020), 1101–1128.
- [2] Yuyang Dong, Hanxiong Chen, and Hiroyuki Kitagawa. 2019. Continuous search on dynamic spatial keyword objects. In *ICDE*. 1578–1581.
- [3] Yuyang Dong, Chuan Xiao, Hanxiong Chen, Jeffrey Xu Yu, Kunihiko Takeoka, Masafumi Oyamada, and Hiroyuki Kitagawa. 2021. Continuous top-k spatial-keyword search on dynamic objects. *VLDBJ* 30, 2 (2021), 141–161.
- [4] Dawei Gao, Yongxin Tong, Jieying She, Tianshu Song, Lei Chen, and Ke Xu. 2017. Top-k team recommendation and its variants in spatial crowdsourcing. *DSE* 2 (2017), 136–150.
- [5] Yafei Li, Lei Gao, Haobo Sun, Huiling Li, and Qingshun Wu. 2022. PRID: An Efficient Pub/Sub Ride Hitching System. In *CIKM*. 4921–4925.
- [6] Yafei Li, Hongyan Gu, Rui Chen, Jianliang Xu, Shangwei Guo, Junxiao Xue, and Mingliang Xu. 2023. Efficient Top-k Matching for Publish/Subscribe Ride Hitching. *TKDE* 35, 4 (2023), 3808–3821.
- [7] Kyriakos Mouratidis, Spiridon Bakiras, and Dimitris Papadias. 2006. Continuous monitoring of top-k queries over sliding windows. In *SIGMOD*. 635–646.
- [8] Shunya Nishio, Daichi Amagata, and Takahiro Hara. 2022. Lamps: Location-aware moving top-k pub/sub. *TKDE* 34, 1 (2022), 352–364.
- [9] Krešimir Pripuzić, Ivana Podnar Zarko, and Karl Aberer. 2015. Time- and space-efficient sliding window top-k query processing. *TODS* 40, 1 (2015), 1–44.
- [10] Xiang Wang, Wenjie Zhang, Ying Zhang, Xuemin Lin, and Zengfeng Huang. 2017. Top-k spatial-keyword publish/subscribe over sliding window. *VLDBJ* 26, 3 (2017), 301–326.
- [11] Xiang Wang, Ying Zhang, Wenjie Zhang, Xuemin Lin, and Zengfeng Huang. 2016. SKYPE: top-k spatial-keyword publish/subscribe over sliding window. *PVLDB* 9, 7 (2016), 588–599.
- [12] Ke Yi, Hai Yu, Jun Yang, Gangqiang Xia, and Yuguo Chen. 2003. Efficient maintenance of materialized top-k views. In *ICDE*. 189–200.