ABSTRACT

Data caching is an important aspect of every software application. Traditional data caching methods implement lookup-based caching mechanisms. These methods avoid the re-computation of data for repeated data accesses. Such techniques fail to analyze the data access patterns and incorporate that information. This paper presents a novel variant of an emerging caching technique - predictive caching. The caching technique introduced here works for a multi-user software application.

The proposed caching method analyzes user footprint data collected by the app. User footprint data is the data about users' access patterns i.e. 'which user accesses which resource at what time'. Using this footprint data, user-trends are generated to befit the collected data. The trend is then used to make predictions about the user's future accesses i.e. `which resource a user would access next`. By predicting a user's future resource-accesses, the computation necessary to generate these resources could be carried out well in advance and the prepared resource can be cached till the time the user actually requests it. In this way, the technique generates two important results; Firstly, it reduces turn-around time dramatically by preparing to-be-accessed resources in advance; Secondly, it improves the cache hit rate by analyzing user footprint data and generating trends. Part of the scope of this thesis also includes designing a framework so that any multi-user application could use this novel caching technique.

Keywords: data caching, predictive cache, user access patterns, pattern mining

List Of Abbreviations and Acronyms

GSP   Generalized Sequential Pattern

TOD   Time of the Day

DOW   Day of the Week

Tech-stack   Technology stack/ecosystem

Chapter 1.   Introduction

1.1.   Overview

Nowadays most of the software applications are shifting towards the cloud-based, multiuser model. Although this solves a myriad of problems associated with the distribution-based model, it introduces new challenges like scalability, latency, performance, etc. With the increasing user-base over time, the application starts experiencing bottleneck issues while serving a large number of users. Popular performance improvement techniques are used to work around such problems. Caching is one of such performance improvement techniques. Traditional caching techniques are lookup-based i.e. they use hash tables to store frequently accessed data in the form of key-value pairs. Upon receiving repeated requests for such data, they just perform a plain lookup and return the cached data if cache-hit occurs and re-compute the data if cache-miss happens. Although very simple to implement, such techniques are not that powerful when it comes to increasing the cache-hit rate.

To increase the cache-hit rate further, we need to analyze the patterns of the user-requests. To understand the previous statement more clearly, let's discuss a scenario. Consider an application that has a professional user base i.e. users working for a firm that uses this application. A typical user uses the application in the following manner: User logs in into the application in the morning and visits a set of dashboard pages, then does nothing till the afternoon; In the afternoon the user accesses configuration pages; Finally, in the evening she again checks out dashboard pages before leaving for the day. This pattern is repeated over the weekdays and no activity has been observed over the weekend.

Now, if we record that user's access patterns (from now on we call it `user footprint`) and analyze the collected data to come up with some sort trend, then we would be able to tell in advance that the user would access *page-Y* after she accesses *page-X*. Having this insight would enable us to proactively load the *page-Y*'s data in advance when the user is accessing *page-X*, even before she actually requests *page-Y.* This would dramatically bring down the turnaround time for user requests and based on the quality of user-trend generated the hit-rate or accuracy of the model would certainly be improved.

One thing worth mentioning here is that resources can be fetched ahead-of-time only if their computation does not depend on any sort of user-provided data or previously accessed resources. For instance, in the above example, we could proactively load *page-Y*'s data only if the data generation process has no dependency either on the user's action or on the currently loaded *page-X*. We term such independently computable resources as "predictively cachable resource". The caching technique being discussed here is restricted to these types of resources.

After an informal overview of the problem statement, in the section, we will formally state the problem statement.

1.2.   Problem Statement

The proposal aims at devising a fully functional framework, which would collect the footprint data, analyze it to generate the trends, and use the trends to perform predictive caching. The predictive caching mentioned here involves anticipating the next potential data-resource which could be accessed and preparing that data-resource before the user makes a request for it.

Following research objectives formally summarize the problem statement:
- Setup a theoretical framework to define and analyze the aforementioned problem
- Define data requirements and design mechanisms to collect user footprint data
- Develop online data mining and machine learning models to generate users-specific system-access trends
- Develop an application-framework which performs proactive data caching based on the generated user trends
- Carry out experimentation to assess model performance

Chapter 2. Problem Analysis

In chapter 1, a gentle attempt was made to introduce the reader to the problem statement.   This chapter is devoted to defining the predictive caching problem as an optimization problem. The motive behind doing so is to come up with some sort of theoretical metric/criterion, which would provide a way to assess the efficacy of the proposed solution. The analysis is broken into two sections, one devoted to defining the problem mathematically, whereas other attempts to strip the problem and extract meaningful insights from it.

2.1.   Problem Description

Before attempting to express the problem mathematically, we shall try to reason why this is an optimization problem. Following narration help answer the question:

"Suppose a software application is using the predictive caching framework. It means the application can use user-specific access trends to predict the next resource to-be accessed by the user. From that user's trend information, the application decides whether it could predict the next resource. If it could predict the next resource, then the application computes the next resource and caches the result. Now the user makes her next request and there are two possibilities here, either the resource requested by her next request is same as the predicted resource or it is different."

Although stated in a very informal language, the above narration summarizes the tradeoffs that we see in a typical optimization problem.

Firstly, let us consider the sentence- "the application decides whether it could predict the next resource", here the application must rely on the trend information to make this decision. A good trend would help application make correct decisions regarding whether it could predict the next resource or not. On the contrary, a misfit trend would make the application misjudge if it could predict the next, to-be accessed resource.

Secondly, "user makes her next request and there are two possibilities…" emphasizes that there is a cost to the mispredictions made by the application. In a positive path scenario, the application correctly predicts that it should cache a certain resource in advance, since the user is likely to request the resource in her next request, and the user does the same. However, in a negative path scenario, the user requests a different resource than the one which the application had predicted and cached. In a neutral scenario, the application did not predict the next resource and hence the question of correct prediction never arises.

From this discussion, it is apparent that in order for the application to perform better at predictive caching, it should maximize positive path scenarios and minimize negative path cases. One could think of hitting a positive path scenario as gaining reward-points and hitting

a negative path scenario as incurring a penalty. In reality, the rewards come from the fact that the user would be getting served on his next request with minimum latency, whereas the penalty comes from the idea that there is a wasted computation in caching the incorrectly predicted resource, which turned out to be of no use.

In the next section, we try to define the problem mathematically as we are acquainted with the necessary background knowledge presented in this section.

2.2. Problem Definition and Analysis

Consider a software application which is using a predictive caching technique. The application has served N requests in a stipulated time window. For simplicity, let's assume that all those N requests were predictively-cachable i.e. the resources requested by those requests could have been computed and cached ahead of time. Of these N requests, for P requests the application predicted that it should compute the next resource and cache the result. Finally, the application recorded that it has correctly predicted H requests. Let's briefly summarize the terminology-

N: Total predictively cachable requests served

P: Requests for which application decided to predict the next resource

H: Correctly predicted requests, cache hit requests

As mentioned in the last section, we should include reward and penalty mechanisms into our problem definition. We chose to define the problem as a penalty minimization problem rather than a reward maximization problem. The penalty could be expressed in terms of two cost factors. Firstly, the cost of a cache miss, the penalty for not been able to reduce request-latency either because of missed prediction or due to incorrect prediction, is assumed to be C_{miss}. Secondly, the cost of incorrect(wasted) prediction C_{waste} incurred by the application when it predicts the next resource, computes it and caches its data, only to see that it made an incorrect prediction. Let us summarize the cost factors-

C_{miss} : Cost of cache miss per request. Penalizing for latency induced

C_{waste}: Cost of incorrect(wasted) prediction per request. Penalizing for wasted computation

After defining the variables and cost factors, we could leverage the relationship between them to come with a total cost function.

- C_{miss} is the cost incurred by each of the *(N - H)* requests, as those many requests have experienced cache miss
- C_{waste} is the cost incurred by each of the *(P - H)* requests, as the application made incorrect next-resource predictions for those many requests.

Combining these two costs linearly yields following total cost function-

$$C_{total}(H, P) = (N - H) * C_{miss} + (P - H) * C_{waste}..(I)$$

Equation (I) gives us the total penalty cost in terms of two variables H and P, two cost factors C_{miss} and C_{waste}, and a constant N. It will serve as the cost function of this optimization problem.

We should take note of the relationship between N, P, and H, at this point. H can't exceed beyond P, as H represents correct predictions whereas P denotes total predictions and total predictions are always greater than or equal to correct predictions. Same way P cannot exceed beyond N.

i.e. $N >= P >= H$ ....(II)

Taking cost function defined by equation (I) and constraints expressed in equation (II), we could define the following optimization problem-

Minimize $Z(H, P) = (N - H) * C_{miss} + (P - H) * C_{waste}$

Subject to

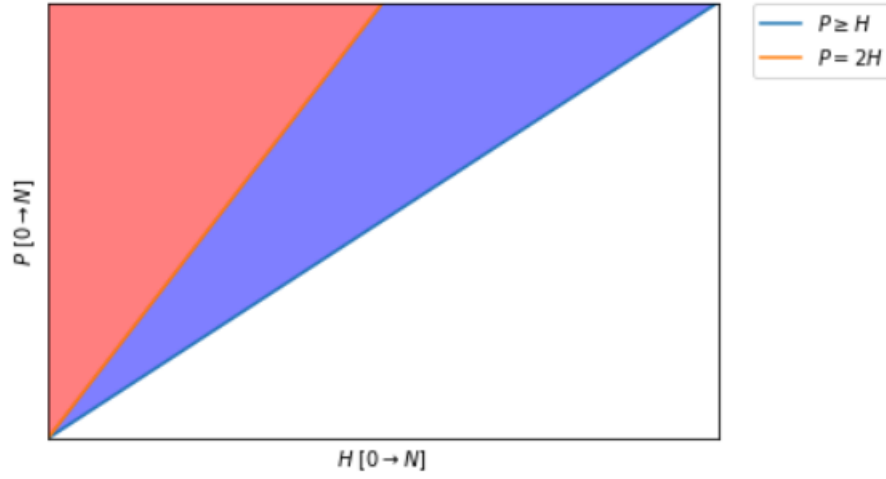$H, P, N, C_{miss,} C_{waste} >= 0$

$N - P >= 0$

$P - H >= 0$

Having framed the optimization problem, we have devised assessment criteria to assess models' accuracy and efficacy. We can use the total cost function as our criterion to compare caching performance across different models, subject to the same data conditions.

We shall explore more on this in the upcoming chapters. In the final section on the theoretical analysis of the problem, we will dive into the bound analysis of variables H and P.

2.3. Bound Analysis

In this section, we shall explore the effects of tweaking variables H and P on the total cost function. The primary motive behind doing such analysis is to gain better insight into the relationship between H, P, and the cost-function. This insight would become useful while interpreting the experimentation results.

Rearrangement of the total cost function makes it take the following form-

$$C_{total}(H, P) = N * C_{miss} + P * C_{waste} - H * (C_{miss} + C_{waste})... (III)$$

Figure 2.1 *Plot showing feasible region (shaded blue and red) and region of efficacy (shaded blue). The region of efficacy is calculated assuming $C_{miss} = C_{waste}$.*

A plot of the cost function shown in figure 2.1. From equation (III) and its plot, we can infer the following insights-

i. Each of the N requests is pre-assigned a unit of miss-penalty cost i.e. C_{miss}. Similarly, each of the P requests carries a pre-assigned one unit of waste-penalty cost C_{waste}.

ii. If no prediction is correct, the total cost becomes $N * C_{miss} + P * C_{waste}$. Each correct prediction brings down the total cost by $(C_{miss} + C_{waste})$ units. The cost reduction could be perceived as the reward gained by the application for a correct prediction.

iii. If the application predicts nothing i.e. P = 0, then the total cost becomes N * C_{miss}.

iv. We declare a predictive caching model as useful if it yields better performance than the model which implements no caching i.e. $C_{total}(H, P) < N * C_{miss}$.

   o Assuming $C_{miss} = C_{waste}$ and solving the inequality yields following relationship $P < 2 * H$, which means for a model to demonstrate the usefulness, at least half of its predictions should be correct

   o The plot introduces this efficacy constraint and the feasible region reduces to only blue shaded part.

Point (iv) gives us an important criterion to analyze the efficacy of the model, whereas equation (III) provides us a quantitative metric to express the model's performance in terms of penalty cost. Equipped with the model assessment criteria, we can now start discussions on model design and implementation. The next chapters are devoted to that task.

Chapter 3.   Modeling Approach

The previous chapter showed us some theoretical aspects of the problem we are trying to model. In this chapter, we shall move a step ahead and dive into the modeling approach considered. We shall first discuss the data requirements of the model. Post that comes the discussion about model definition. In the end, we conclude the chapter after a brief discussion on the model assessment topic.

3.1.   Data Requirements

Naturally, for any data mining and machine learning modeling problem, one must first identify the data one wants to analyze. The task of modeling the predictive caching problem is no different. As introduced briefly in Chapter 1, we need user footprint data to model the predictive cache framework.

Attribute Name	Attribute Data Type
userId	varchar
resourceId	varchar
accessTime	datetime

Table 3.1 *Table showing mandatory attributes of the user footprint data, along with their data types*

userId	resourceId	accessTime
36277	summary-dashboard	2019-09-12 08:43:51
36277	notification-center	2019-09-12 08:44:11
36277	summary-dashboard	2019-09-13 07:22:56
36277	user-manager	2019-09-13 07:23:20

Table 3.2 *Table showing sample user footprint data, populated with mandatory attributes*

Table 3.1 shows the structure of user footprint data, it specifies that the footprint data must have three mandatory attributes – userId, resourceId, and accessTime. Table 3.2 shows us sample footprint data. We can interpret the first entry in Table 3.2 as- "User *userId(36277)* accessing resource/page *resourceId(summary-dashboard)* at time *accessTime(2019-09-12 02:43:51)*".

Having briefly discussed the data requirements, we can move ahead to the task of defining the Access-Trend Generation model. The model uses this data to generate user-

specific trends.

3.2.   Access-Trend Generation Model

Access-Trend Generation modeling aims at creating a data mining and machine learning model, which feeds on the user footprint data to create user-specific access trends. We will walk through the approach taken for creating this model.

The primary philosophy behind the modeling is to consider each access-entry in the footprint data as a "transaction". That will turn the user footprint data into a collection of transactions per user. After this initial preprocessing, the remainder of the model pipeline works on the per-user transaction group. Figure 3.1 shows the pictorial representation of the splitting of user footprint data into per-user transaction groups.
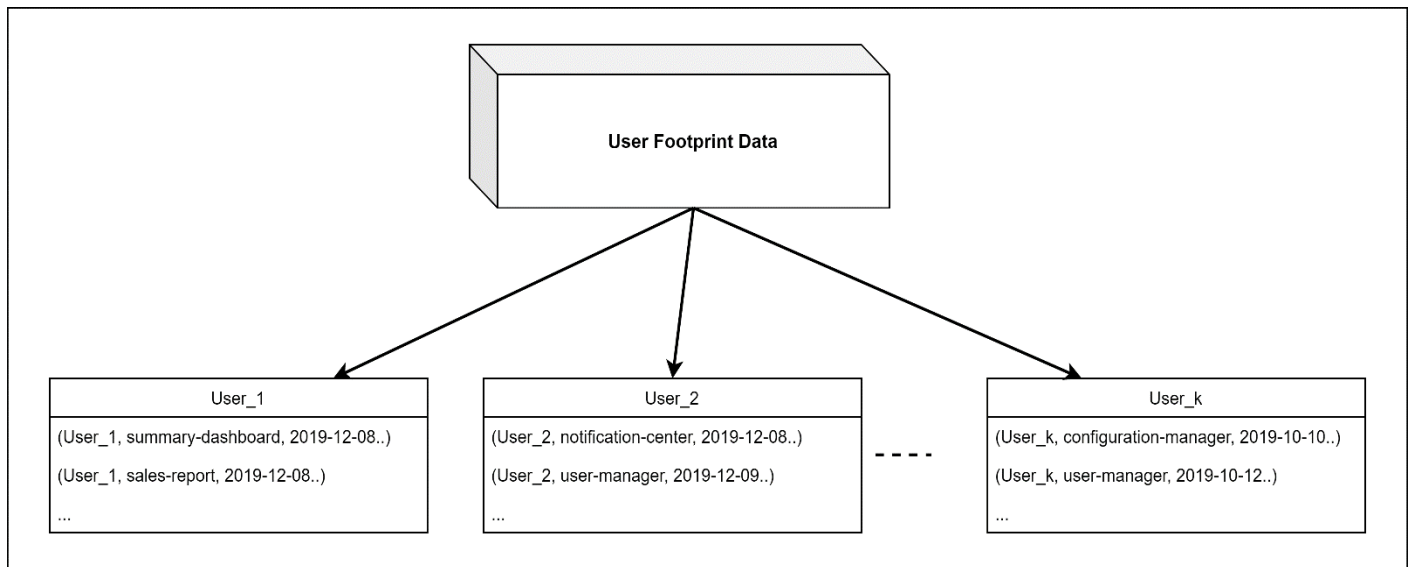
Figure 3.1 *Grouping the user footprint data into per-user transaction group, and filtering out those transaction groups which contain fewer transactions than the required threshold*

Once the per-user transaction group is generated, it is again divided into multiple transaction sets, based on the temporal distance measured using accessTime attribute i.e. two transactions with closer accessTime would be put into the same transaction set. The idea behind this splitting scheme is to find out the user's access burst i.e. the set of resources she accessed one after the other such that each next resource is accessed no later than some stipulated cool-off time-period "T". Figure 3.2 shows an example of how this sort of splitting logic works. Once this splitting scheme is applied to the per-user transaction group we are left with what we can call "a per-user group of transaction-sets".

A sequential pattern mining algorithm, GSP, is then run on the group of transaction-sets. The GSP outputs repeating sequences across transaction sets, which satisfy provided support-threshold. These repeating sequences outputted by the GSP algorithm could be called raw access-trends of that user. We are calling it "raw" because it completely lacks the temporal
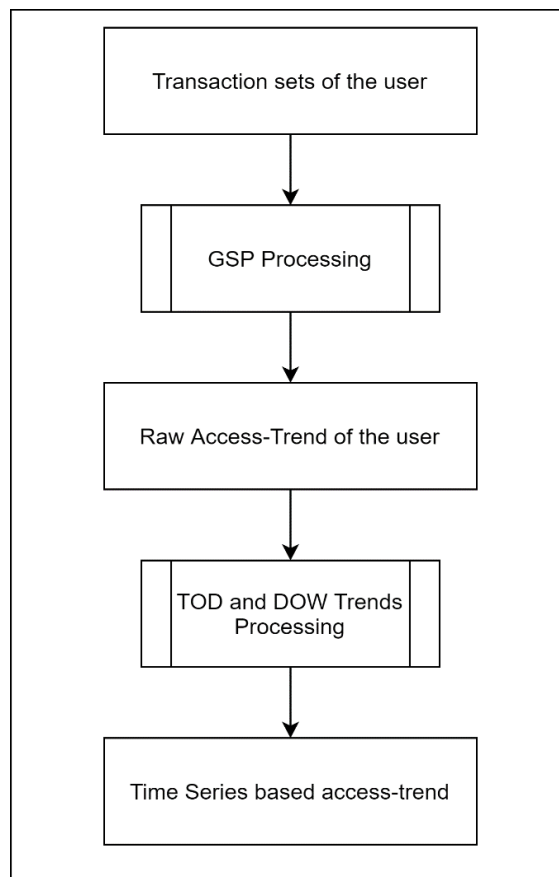
Figure 3.2 *Splitting the transaction group of User_1 into multiple transaction sets based on accessTime temporal distance. Time_diff value calculated above is the difference between two consecutive transactions(sorted by accessTime). Here, the master transaction set is split into two transaction-sets based on the cool-off time-period "T=5 minutes".*

aspect i.e. the raw-trend does not provide any information regarding TOD(time of the day) or DOW(day of the week) time-series patterns. We need to process these raw-trends, further down the model-pipeline to create time-series trends.

Raw access-trends give us helpful insights about what sequence of pages a user accesses again and again. This insight is the crux of the predictive caching model, as this information about the user's access sequence would help the caching framework predict the user's next likely access. Although very useful, the raw access-trend has no information about the time of the occurrence of this access sequence. Because it lacks this very useful information, we need to further process the raw-trends to come up with the time-series trends like TOD pattern(for instance, a user consistently accesses 'sales-report' followed by 'summary-dashboard' in the morning) or DOW pattern(for example, a user repeatedly accesses 'configuration-manager' followed by 'data-import' screen on Mondays). We can comb through the transaction data and use raw-trends to generate this sort of furnished trend information.

Figure 3.3 summarizes the trend generation pipeline. On top, we have the transaction set outputted in Figure 3.2. Those have been consumed by GSP processing to create raw trends. These raw access-trends, in turn, are churned by TOD and DOW processor to create furnished time-series based access-trends. Figure 3.4 provides a sample of the generated time-series trend and describes its structure.

After the discussion about user-trend generation, it's time to look into how these trends will be getting used. In the next section, we'll take a brief tour of the trend-matching model.

Figure 3.3 *Flowchart showing trend generation pipeline.*

Figure 3.4 *Sample of a user trend. The user "35265" exhibits five pattern-trends, each of which is characterized by a repeating access sequence and time-series trend information. Shown above is the pattern-trend for the access-sequence [summary-dashboard → sales-report]. The time-series trend is a two-level trend with nesting of TOD information inside DOW. For instance, the count against the "hourRange_6am_to_9am" field is the number of occurrences of the access-sequence [summary-dashboard → sales-report] between 6am and 9am on Mondays.*

3.3.   Access-Trend Matching Model

The matching model deals with the consumption of the generated user trends. It serves as a prediction module of the caching framework. The flowchart shown in figure 3.5 summarizes the working of the access-trend matching model. The chart runs the reader through a hypothetical example to help illustrate the internal working of the trend matching model.
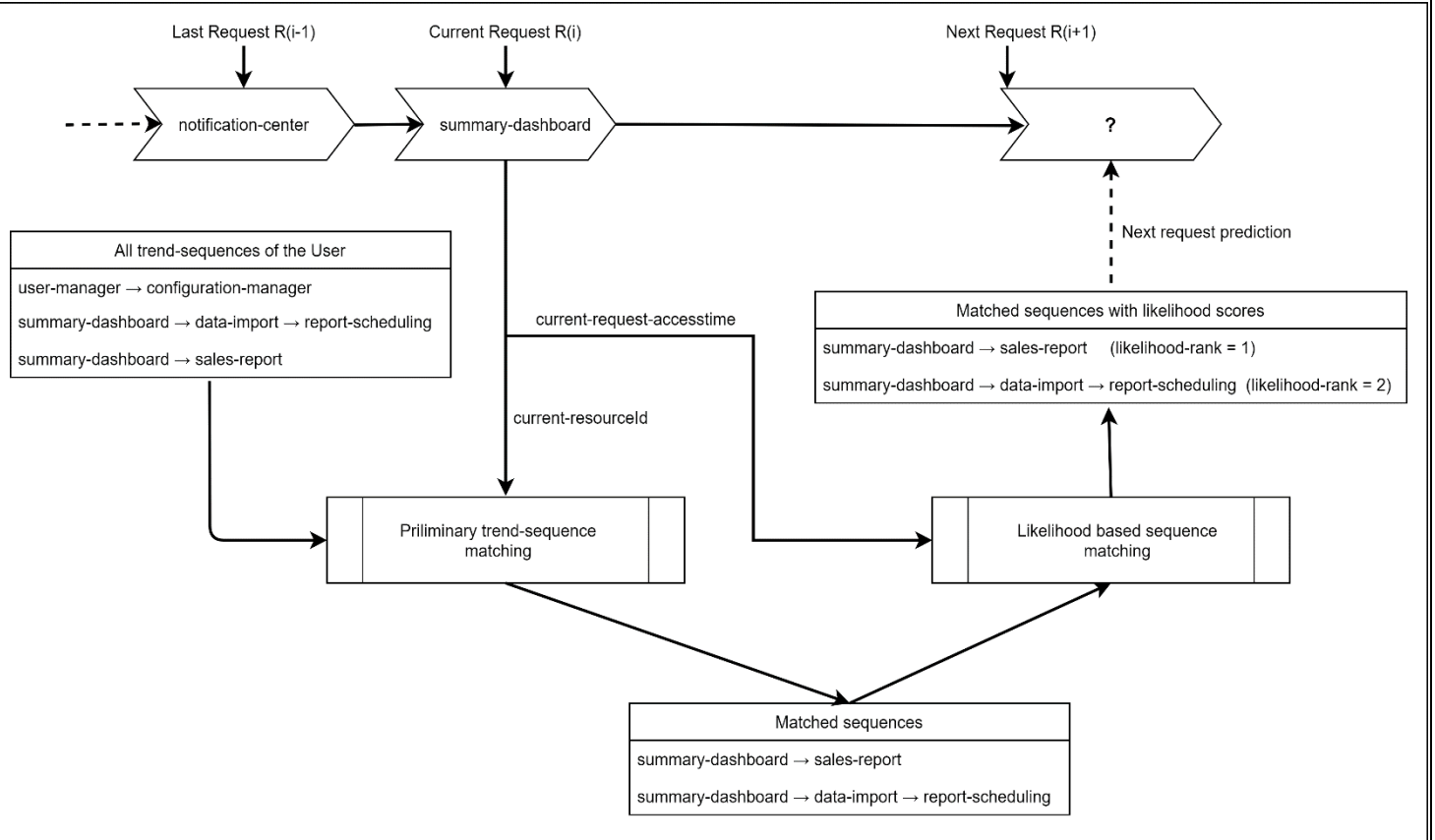
Figure 3.5 *Flowchart showing trend matching pipeline. The matching process is carried out for each incoming request. The output of the matching process is either the predicted next-request or null.*

As shown in the flowchart, the matching processing starts on the arrival of each user-request. The goal of the trend-matching is to predict the next user-request. First of all, all user trends are searched for by a preliminary sequence matcher to filter out irrelevant sequences. In the example described in figure 3.5, initially, there were 3 trend-sequences present for the user. After the action of the preliminary-matcher, the trend-sequence [user-manager → configuration-manager] is removed from the search-space. The elimination stems from the fact that the current request is for resource "summary-dashboard" and the relevant trend-sequences would be the ones that contain a reference to the current resource. Once the eliminations applied by the preliminary trend-sequence matcher, we are left with two potential trend-sequences.

Since both the trend-sequences contain a reference to the current resource, we need some other criterion to choose between them. The trend-matching model uses previously generated time-series information associated with the trend-sequences, to determine their

likelihood of occurrence in the "current context". "Current context" is the information about the access-time of current-request. Naturally, the trend-matching model scores those sequences higher which have a higher frequency of occurrence in current time-setting (TOD and DOW pattern). Once the likelihood scores are assigned, then the sequence which satisfies likelihood-threshold criteria and possesses the highest likelihood score is chosen as the prediction result.

Most of the time, there is a null matching result. That means the trend-matching model cannot predict the next request. If the prediction is non-null, then the resource associated with the predicted next request is computed and the results are stored, ready to serve.

The purpose of this section was to introduce the reader to the approach used by the trend matching model to consume the generated access-trend and make predictions. Now it's time to take a tour of the wholistic caching framework, in terms of its architecture and implementation details. The next chapter is added with that purpose in mind.

Chapter 4.   Caching Framework

This chapter revolves around the discussion about the caching framework- its modular architecture and implementation technicalities. The chapter starts off by providing an architectural overview of the framework, wherein it engages in a brief discussion about the framework modules. Once the reader is familiarized with the high-level framework architecture, the commentary about the framework implementation details helps shed some light on the nitty-gritty of realizing the architecture.

4.1.   Framework Architecture

Primarily, the framework is broken into three modules, analytics-module, online-prediction-module, and configuration-module. The analytics module is equipped with the ability to generate user access-trends from user foot-print data. The Access-Trend Generation Model discussed in Section 3.2 is implemented in the analytics module. On the other hand, the online-prediction-module does the tasks of Trend Matching, as discussed in Section 3.3. Online-prediction-module also performs core tasks such as resource-caching, user footprint data collection. The last major module of the framework, configuration-module does the work of recording hyper-parameter configuration and orchestrating the trend-calibration process.
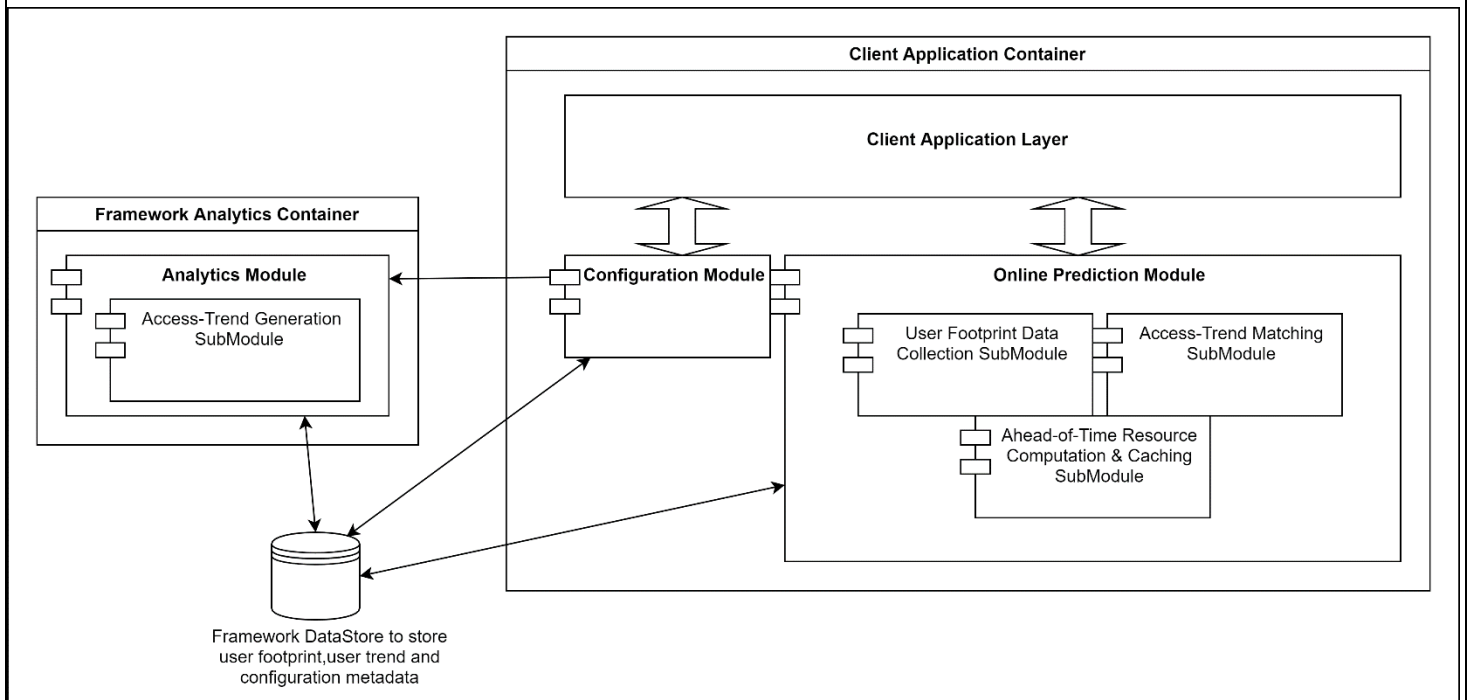
Figure 4.1 *Architectural overview of the caching framework. Various framework modules, submodules are shown, along with the information about the interaction between them. Also shown is a sample containerization approach, with a separate container for analytics module and online-prediction-module running inside client-app container along with the configuration module.*

As shown in figure 4.1, one could see the modules and submodules of the caching framework. First of all, we could see that the framework modules are distributed across two

containers- configuration-module and online-prediction module residing with the client application, whereas the analytics module running inside a separate container. This is not a mandatory requirement, except for the constraint that the online-prediction module must reside inside the client-application container. The tight coupling between the online-prediction module and client-application is because of the Ahead-of-time-resource-computation & caching module. We shall revisit this specific submodule shortly in the next section.

The architecture highlights one thing, that there is a common framework datastore shared by all the modules. All of the data communication and some of the control communication between the modules happens through the shared data store. The datastore primarily stores user footprint data, generated access-trends, and hyperparameter values. Another thing to notice amongst the interaction shown in figure 4.1 is the communication link between configuration-module and analytics-module. The configuration-module orchestrates trend-generation processes happening inside analytics-module.

After the brief overview, the discussion can be moved forward to include implementation details and challenges. The next section summarizes the implementation challenges faced while realizing the architecture. Although a somewhat subjective matter, it is still worth bringing up.

4.2.   Implementation Details

This section summarizes the technologies used for implementing the caching framework. Then comes the discussion about important behavioral aspects of the online-prediction module, especially the "ahead-of-time resource computation & caching" submodule. Since most of the communication with the client application is done by this submodule, we will discuss in brief its internal workings. And finally, we glance over the challenges faced while realizing the framework.

Module	Technology stack
Configuration Module	Java, Spring
Online-Prediction Module	
Client Application	
Analytics Module	Python
Framework Datastore	MongoDB

Table 4.1 *Information about technology stack used for implementing each of the primary modules.*

The technology stacks used for implementing different modules are listed in table 4.1. As discussed earlier, owing to the internal working of the "ahead-of-time resource computation & caching" submodule, it should share the same tech-stack as the client application. Let's try

to understand the reasons behind the tight coupling between the client application and the submodule.

As a part of the initialization process, the client application is expected to register its resources into the framework's resource-registry. The resource-registry maintains mappings of the form [resourceId → resource-supplier]. The resource-supplier is a functional interface which when called, performs computation and yields data of that resource.

In section 3.3, the flow of the access-trend matching model has been studied. The trend matching module predicts the next resource, which is likely to be accessed. Once this information is available, then the "ahead-of-time resource computation & caching" submodule uses mappings from resource-registry to get the supplier, executes the supplier, and caches the yielded data. All this happens in parallel to the current request thread, in a separate thread. This resource execution flow tightly couples the submodule with the client application.

Figure 4.2 summarizes ahead-of-time execution flow, just discussed. In addition to that, the schematic also shows what happens when the next request comes in. When the actual next user request comes in, the submodule checks if the request is for the resource which it had cached earlier. If yes then the cached data is returned thus avoiding the computation, else the cached data is discarded and the requested resource is computed as usual.
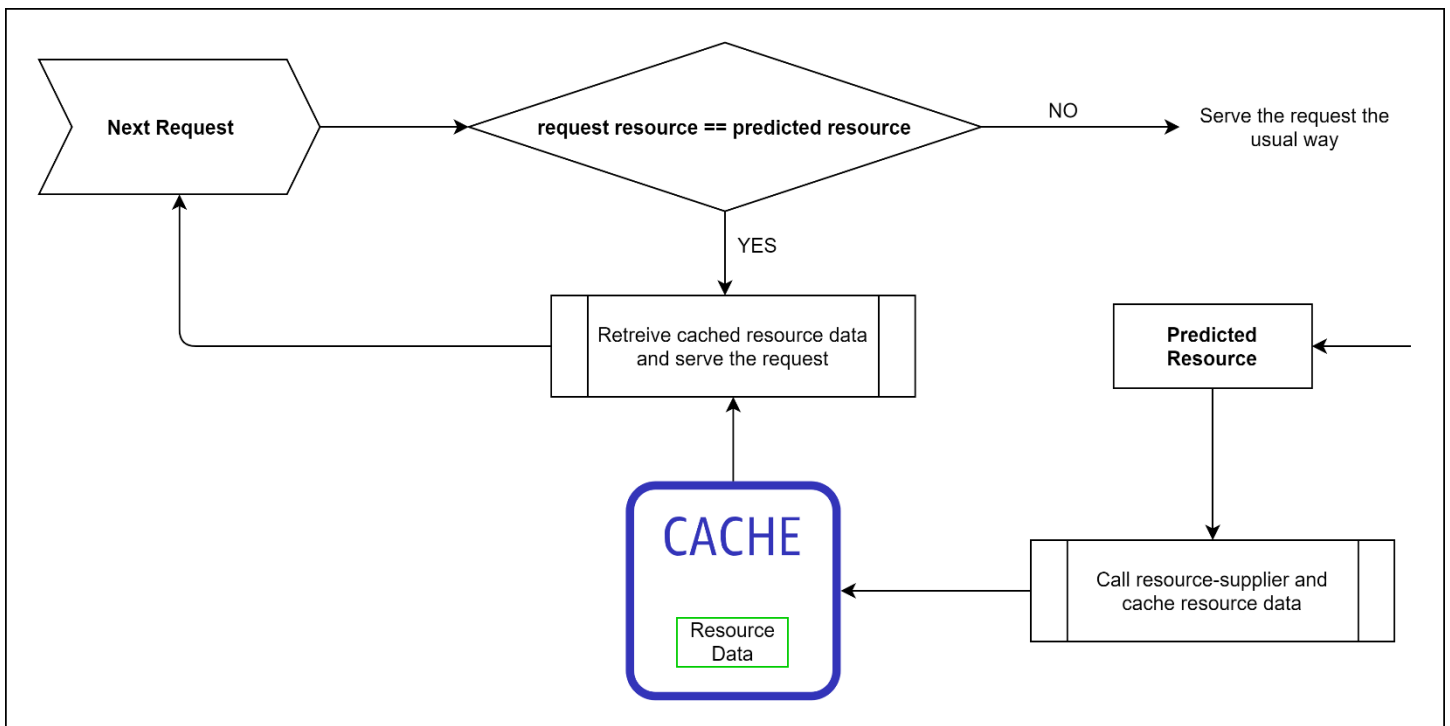
Figure 4.2 *Flow diagram of the sequence of operation happening after the actual next request comes in. Even before this moment(in an ideal scenario), the "ahead-of-time resource computation & caching" submodule takes predicted resource value from the trend-matching module, executes the resource, and caches the resource-data. On arrival of the next request, the submodule simply evaluates the condition and decides whether to intervene and serve the request or let the application handle the request.*

Finally, we are ready to discuss the challenges posed by the architecture and its implementation. First of all, the tight coupling of the "ahead-of-time resource computation & caching" submodule makes the framework difficult to port to the new tech-stack, without being partly rewritten. Second, the language-specific dependency of resource-registry i.e. a language in which the client application is written must support functional paradigm features like suppliers. Last, the dependency on the runtime environment for parallel processing makes the framework resource intensive. This challenge can be addressed by ingeniously implementing the online-prediction-module to make its resource-footprint as low as possible.

After wrapping up the discussion on the caching framework's architecture and implementation we shall move ahead to look into the experimentation part. The next chapter provides a platform for the discussion of experimentation setup and results.
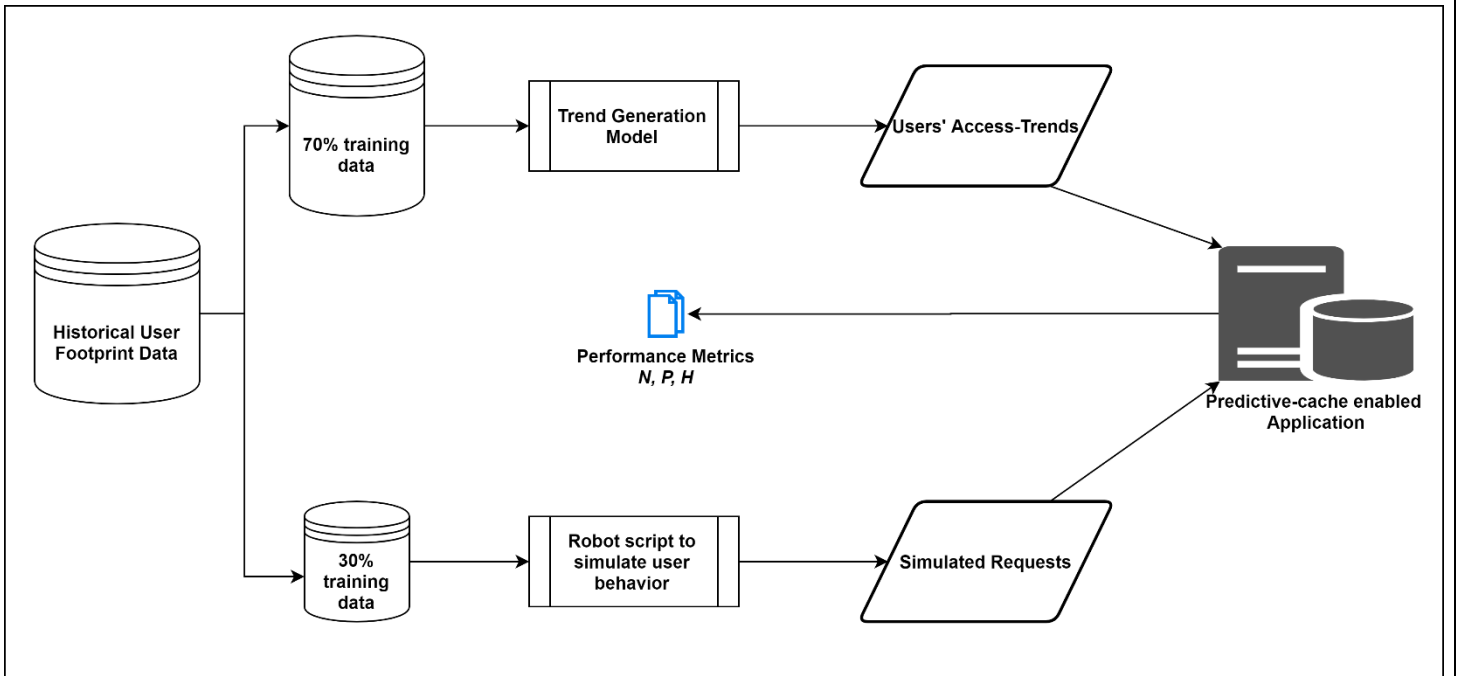
Chapter 5.   Experimentation

So far the reader has been introduced to all prime aspects of the predictive caching framework; from theoretical analysis to implementation. It is time to feed some data into the developed system and analyze the experimentation results. This chapter focuses on the discussion about the experimentation methodology.

5.1.   Methodology

Before discussing the methodology, let's first understand the data collection strategy that has been employed for this project. Sample user footprint data is collected from a live production system. All of the data is of historical type i.e. we have collected the user footprint of some sample users from the last 6 months. The data is split into two sets- 70% of the data constitutes training-set and 30% constitutes the test-set.

Using the training data the user trends are generated, whereas the testing data is used to simulate users' system usage behavior. The testing data is reverse engineered to create user-requests from the user-footprint. A robot script is created which sends the requests to a predictive cache-enabled application. The application treats these requests as regular user-initiated requests. The framework records its performance metrics i.e. $N, P,$ and H values discussed in chapter 2. Figure 5.1 summarizes this process.

Figure 5.1 *Schematic representation of experimentation methodology. The aim is to generate performance metrics N, P, H for each variant of the model.*

Once the robot script finishes sending all test-requests to the application, a recorded performance metrics' dump is taken. These metric values *viz. N, P, H* are then subjected to model assessment criteria discussed in chapter 2. This process is repeated for various variants

of the "access-trend generation model". The aim is to assess these models' relative performance for the current dataset. The next section carries out this assessment in detail.

Bibliography and References

[1] R. Srikant and R. Agrawal. 1996. Mining Sequential Patterns: Generalizations and Performance Improvements. In Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology (EDBT '96), Peter M. G. Apers, Mokrane Bouzeghoub, and Georges Gardarin (Eds.). Springer-Verlag, London, UK, UK, 3-17.

[2] Bontempi G., Ben Taieb S., Le Borgne YA. (2013) Machine Learning Strategies for Time Series Forecasting. In: Aufaure MA., Zimányi E. (eds) Business Intelligence. eBISS 2012. Lecture Notes in Business Information Processing, vol 138. Springer, Berlin, Heidelberg

[3] S. Parthasarathy, M. J. Zaki, M. Ogihara, and S. Dwarkadas. 1999. Incremental and interactive sequence mining. In Proceedings of the eighth international conference on Information and knowledge management (CIKM '99). Association for Computing Machinery, New York, NY, USA, 251–258. DOI:https://doi.org/10.1145/319950.320010

[4] H. Conrad Cunningham, Yi Liu, Cuihua Zhang, Using classic problems to teach Java framework design, Science of Computer Programming, Volume 59, Issues 1–2, 2006, Pages 147-169, ISSN 0167-6423, https://doi.org/10.1016/j.scico.2005.07.009

[5] Brad Glasbergen, Michael Abebe, Khuzaima Daudjee, Scott Foggo, and Anil Pacaci. 2018. Apollo: Learning Query Correlations for Predictive Caching in Geo-Distributed Systems. In Proceedings of the 21th International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018.

[6] E. Rappos and S. Robert, "Predictive caching in computer grids, " in Proceedings of the 2013 13th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, ser. CCGrid 2013. Delft, Netherlands: IEEE Computer Society, May 2013.

[7] Mohan, C.. (2001). Caching Technologies for Web Applications.