A Cloud Native Framework for Real-time Pricing in e-Commerce

Archana Kumari[1], Mohan Kumar. S[2]
School of Engineering and Technology, CMR University, Bengaluru, India[1]
Directorate of Research and Innovation, CMR University, Bengaluru, India[2]

***Abstract*—Real-time pricing is a form of 'dynamic pricing,' and it enables online sellers to adjust prices in real-time in response to variations in demand and competition to achieve higher revenue or improve customer satisfaction. As modern e-commerce implementations become more cloud-based, this paper proposes a cloud-native framework for a real-time pricing system. We take a requirement driven approach to come up with a modular architecture and a set of reusable components for real-time pricing. Following DSRM methodology, during the design phase, we identify and develop the theoretical foundations for key parts of the system, such as pricing models, competition and demand watchers, and other analytics components that fulfill the functional requirements of the system. At the stage of implementation, we describe how each of these components and the entire cloud application will be configured using an AWS cloud native implementation. As a framework, this work can support a variety of pricing models, demonstrating that multiple pricing models have been discussed. Other low-latency, reusable components described in this work provide the ability to react quickly to changes in demand and competition. We also provide a price-cache that decouples pricing model calculation from end-user price requests and keeps price query latency to a minimum. For a real-time system, where latency stands to be the most desired NFR, we validate the system for price-request latency (found to be a single digit of milliseconds) and market reaction latency (less than a second). Overall, our proposed framework provides a comprehensive solution for real-time pricing, which can be adapted to different business needs and can help online sellers optimize their pricing strategies.**

Keywords*—*Real-time pricing; cloud-native design; system-design; pricing-framework; Amazon web services

I. INTRODUCTION

e-Commerce has become increasingly competitive; therefore, enterprises that intend to win must be able to offer their products and services at prices that are competitive with those of their rivals. Pricing is a crucial aspect of e-commerce, and proper pricing can help to attract and retain customers, increase sales and revenue, and optimize profits [1]. The term "dynamic pricing" refers to the practice of adjusting prices in response to fluctuations in supply, demand, and other factors in the market. Profits can be maximized, sales increased, customer happiness improved, and market share preserved by employing dynamic pricing [2]. Firms are looking to provide prices that are more personalized to each client by using a customer's location and purchase history, among other factors. Dynamic pricing is popular in industries other than e-commerce, including energy, transportation, and financial markets [3].

Real-time pricing is a form of dynamic pricing that involves adjusting prices in response to market factors as they take place in real time. Real-time pricing allows businesses to respond to market conditions in a near-instantaneous fashion, which can help them optimize their profits and minimize risk. By using real-time data, businesses can make more informed pricing decisions and ensure that they are not charging too much or too little for their goods or services [4].

Factors of Dynamic Pricing: Demand is the most important factor in dynamic pricing; when there is high demand, a company may be able to charge a higher price, and in the event of low demand, the company may need to lower the price in order to attract buyers. The amount of competition and what firms know about their prices can help them decide on pricing strategies like undercutting, matching, leading, or skimming. A company may use dynamic pricing to differentiate its prices based on customer segmentation. For example, a company may charge higher prices to customers who are more willing to pay, typically referred to as "myopic customers". Similarly, customer loyalty enables differentiated pricing, i.e., if a company has a loyal customer base, it may be able to charge a higher price for its products or services. Another crucial factor is the level of inventory, i.e., if a company has a limited supply of a product, it may be able to charge a higher price due to the scarcity of the product. Other pricing factors may include seasonality, time of day or week, weather, choice of payment methods, etc. [1] [2].

Cloud technology has undergone significant advancement in recent years, and there are several key developments like containers, serverless computing, AI/ML & IoT etc. The adoption of cloud computing has been growing rapidly in recent years. According to a report by IDC, global spending on cloud services is expected to reach $1.3 trillion in 2025, up from $706 billion in 2021 [5]. Cloud computing is increasingly important for businesses as they look to take advantage of cost savings, increased agility and flexibility, and improved scalability and reliability. A wide range of online businesses leverage cloud infrastructure to host stores, manage transactions, and offer integrations and extensions to customize and optimize their online presence [6]. Cloud native refers to a method of developing and deploying applications that fully utilize the cloud computing paradigm. Two popular cloud-native application development paradigms are: 1) Microservices architecture splits an e-commerce platform into discrete services that may be independently developed, deployed, and scaled [7]. 2) Serverless architecture leverages cloud services such as AWS Lambda to manage infrastructure

and run programs without servers. This decreases expenses and improves company agility [8].

A wide range of e-commerce businesses use cloud computing. Amazon's online business runs almost entirely on AWS [9], a platform that hosts and oversees its online store and other operations. Alibaba utilizes the cloud to power its operations on Alibaba Cloud, which handles and maintains all the company's business operations, including its e-commerce platform [10]. Apart from e-commerce businesses, there are cloud-based e-commerce platforms that host and administer online stores. Shopify [11] and Magento [12] are such popular e-commerce platform for establishing and managing online stores [11]. It leverages cloud infrastructure to host stores and manage transactions, and offers integrations and extensions to customize and optimize its online presence.

Real-time pricing has the potential to improve revenue and give businesses real-time control over prices. However, its implementation poses challenges such as real-time decision-making, managing large volumes of data, achieving scalability, and accommodating diverse product types. Cloud-native technology has become the preferred platform for contemporary e-commerce development, and designing a cloud-native application can help address many of these challenges. To increase adoption of real-time pricing by sellers, it is important to solve the challenge of real-time decision-making and develop a reusable framework that can be used for different e-commerce product types. The objective of this paper is to develop a framework for a real-time pricing system that helps online stores optimize revenues or other business specific pricing requirements. We will begin by identifying the requirements and specifications, using them to guide the design and build the necessary functional foundation. To implement the framework, we will use cloud-native technology, which offers cost-effectiveness, scalability, and quick time-to-market. In addition, we will follow the philosophy of "meeting customers where they are", by implementing the system on a popular cloud platform. Specifically, we have chosen AWS, which is currently the most widely used cloud solution.

The remaining sections of this paper are organized as follows: Section II surveys the existing literature, explores all the different types of work that has been carried out in the field of dynamic pricing. There we will also identify the lack of system design and implementation work in the existing literature, among other things as a research gap. The methodology and system requirements are discussed in Section III and Section IV, respectively. Section V presents the high-level design of the real-time pricing system, while Section VI focuses on the cloud-native implementation of the same. The results of this study are discussed in Section VII, and Section VIII provides the conclusion and potential future work.

II. Literature Review and Research Gap

A. Background of Dynamic Pricing in e-Commerce

Dynamic pricing is a multi-disciplinary research area as it involves the integration of knowledge and techniques from multiple fields, including economics, mathematics, statistics, computer science, and behavioral psychology [13]. The primary goal of dynamic pricing is to adjust prices based on various factors such as supply and demand, competition, and customer behavior, to remain competitive in the market [14]. Dynamic Pricing implementation may involve statistic, algorithms, and machine learning techniques to predict demand and adjust prices accordingly [15]. Dynamic pricing has several benefits, including increased sales, improved profit margins, and better customer satisfaction [16].

B. Algorithms for Dynamic Pricing

1) Rule-based pricing: This is dynamic pricing based on predefined rules and conditions that adjusts prices for goods and services accordingly. In this strategy prices are dynamically determined on factors such as demand, supply, competition, time of day, and other market conditions. [17].

2) Competition pricing: This refers to the practice of basing prices on those offered by competitors. It can be an effective strategy for a low-cost supplier entering a new market. However, blindly following competitive pricing can hinder a company's ability to capitalize on shifting customer perceptions of value and brand differentiation. [18].

3) Linear programming approaches: In order to maximize revenue, profit, or market share, among other business goals, linear programming can be used to optimize pricing decisions over time. Linear programming for dynamic pricing seeks to maximize a predetermined objective function by first developing a pricing model that factors in several factors, including but not limited to customer demand, product availability, and competitive pressure [19][20].

4) Statistical and machine learning approaches: (non) Linear regression can be used in dynamic pricing to model the relationship between the price of the product or service and the demand for it. One can collect data on past sales and prices and use it to train a linear regression model and possibly optimize sales revenues (or profit) based on a parametric model using (non) linear regression [21]. Machine learning algorithms like reinforcement learning (RL, which is typically used to maximize rewards in games) have the potential to solve dynamic pricing problems in various scenarios like varying demand and competitive settings. [22].

C. Cloud Native based Application Development

Cloud-native technology is becoming more popular as businesses seek ways to build and deploy software applications that can handle high traffic, frequent updates, and rapid scaling. Gannon et al. define cloud-native as the practice of developing and deploying highly scalable, resilient, and secure cloud-based software [23]. Cloud has popularized microservices that divides a larger application into smaller, more manageable services that can be built, deployed, and scaled independently. Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications [24]. The current academic literature lists some interesting case studies conducted with cloud native technology. Using technologies such as microservice architecture, event-driven architecture, domain-driven design, and containerization, Pan et al. schedule CPU and GPU resources for their cloud-native online judge system [25]. A cloud-native smart operation management platform

architecture is proposed by Lang et al., which would centralize the technical architecture and data interaction of various operation subsystems [26].

D. Research Gaps

Although several studies have been conducted on pricing strategies for e-commerce, there is a lack of research that specifically addresses real-time pricing systems that can adapt to changing market conditions and customer types in real-time. There is also a lack of studies that examine the technical aspects of implementing real-time pricing systems, such as the selection of appropriate technology platforms, pricing algorithms and building reusable components.

III. Method

For this work, we use DSRM (Design Science Research Methodology), a research paradigm that emphasizes the generation and validation of prescriptive knowledge. Design science is a serious research approach for creating solutions to real-world engineering issues. As contrasted with how things are in the natural sciences, DSRM is concerned with how things ought to be, that is, with creating artifacts to accomplish things. The steps of the DSRM [27] process include problem identification and motivation, specification of the solution's objectives (requirements), design and implementation, demonstration, evaluation, and communication. Algorithms, user interfaces for computers and people, design approaches (including process models), and languages are among the kinds of artifacts for which design science research is often used. For this work, problem identification and motivation for a real-time pricing system have already been established (in the first section), for the next few sections, we are going to cover requirement, design, implementation, result analysis and will provide conclusion.

IV. Requirements

A. Functional Requirement

FR1: A real-time pricing system should provide a suitable price, as per the business need, that optimizes revenue, customer experience or stakeholder's expectations.

FR2: Price should be able to be derived from one or more factors: market factors (demand, competition prices), inventory level, and possibly different customer segments.

B. Non-functional Requirement

NFR1: Configurability/Reusability - Depending on the product type and its current state in the product lifecycle, more than one pricing logic may be configured to run; i.e., the framework should have the ability to switch to different pricing logic.

NFR2: Response latency – Pricing response should be quick, and there should be very minimal deterioration in the price lookup.

NFR3: Reactive Latency – System should react to changes in factors listed in FR2 in an automated fashion within an acceptable time-frame.

NFR-4: Asses the aspects of availability, scalability, and infrastructure cost.

V. Design

This section explains the theoretical design for a real-time pricing system that meets the functional requirement established in the previous section, as well as how it paves a way out for NFRs.

A. Typical Setup of Pricing-module in an e-commerce Application

In a typical e-commerce setup, the user interacts with the catalogue service, which sends them a list of products along with information like the price and the product's description. A product may have a static (fixed) price or be dynamically priced. When there is static pricing, a special database created just for this purpose provides the price data to the catalogue service. When there is dynamic pricing, the prices might need to be calculated on the fly. The catalogue service should contact the pricing service to learn the item's price. Fig. 1 provides an illustration of this variation. Even though dynamic pricing can automate the price generation process, a trivial implementation may result in longer "price-response" latencies and higher computational costs.

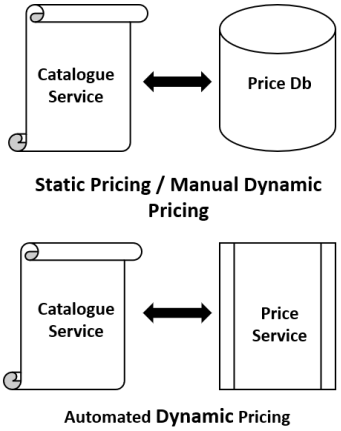

Fig. 1. Static vs dynamic pricing implementation.

B. Pricing Service – High Level Design

The functional requirements (FR1 and FR2) served as our inspiration for the high-level design of the pricing service that is presented below (Fig. 2). The pricing model is in charge of determining the prices. The model may be dependent on inventory, demand, and price competition. By depicting the components in Fig. 2 as distinct parts, we suggest that they can all function concurrently and prefetch/prebuild all the input required by the pricing model. The use of a pricing cache eliminates the need to invoke the pricing model, as it serves cached the pricing results produced by the model to the catalogue service.

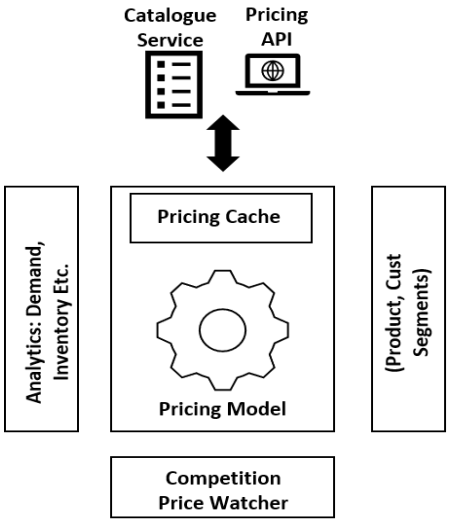

Fig. 2. HLD for real-time pricing system.

C. Component Design

1) Competition watcher: The competition watcher component has the responsibility of routinely monitoring and reporting changes in competition prices. The interval at which the "competition watcher" can detect a change in competition prices will determine how quickly the real-time pricing system responds to market prices. A trivial design for this component would be to poll the market for prices at regular intervals. Better reactive latency (FR3) would result from shorter polling intervals, but at the cost of higher computational costs. An ideal polling frequency would be determined by considering how frequently the competition changes their prices. Let us say a competitor increases the price of her product at a rate of λ, Poisson's theorem can be used to calculate the probability that the price (PCP_t) will change during the period t. (shown in Eq.(1)).

$$PCP_t = 1 - e^{-\lambda t} \quad (1)$$

PCP_t becomes an input configuration parameter for the system for a given competitor; let us call it the desired probability (to detect the competition price change, denoted by DP). This helps figure out the polling interval for which the probability of competition price change is configured.

$$Polling\ Inteval, T = \frac{\ln(\frac{1}{1-DP})}{\lambda} \quad (2)$$

Initially rate λ is unknown and can be trivially calculated by polling at a small period during the initial setup phase. Once λ is known, Polling interval T can be found using Eq. 2, and should be used to poll the prices. Should competition later increase the frequency of price change, it would be seen by the competition watcher that, number of price changes in the interval T, is more than 1/DP, and that would trigger a re-calculation of λ. When there are fewer samples than 1/DP in the competition prices, this means the competition has decreased their price change rate, and a new polling interval can be calculated using Eq. (2) again.

2) Demand watcher: Unlike in competition, where price changes can occur all at once and must be monitored at short intervals, the demand can be calculated once, for a specific time period, such as the previous hour or day. There could be two potential indicators of demand: 1) the price request rate and 2) the item sale rate. However, our understanding is that, sale is also a function of price, and price is variable; thus, in this work, we only consider the first one, i.e., the rate of incoming requests, as an indicator of demand. The architecture influences demand estimation procedure; however, the two most common approaches are:

- *Transaction database query*: Transaction databases are mostly relational databases with transaction time as an index. Using an SQL query, it is trivial for such a database to get the number of transactions in a certain time frame.
- *Transaction stream accumulators*: Another approach is to stream the transaction as it occurs and store the summary in a database table. This method is useful when there is no database from which to obtain information.

3) Other analytics: The pricing models (many of it) need analytics data, for example sale-probability, customer segments, existing inventory (or projection of upcoming replenishment). Most of the analytics do not change very frequently and hence computing them once (say daily) would suffice.

D. Pricing Models

1) Rule base pricing: This style of pricing involves the dynamic adjustment of a product's price in accordance with a predetermined set of rules. Such a rule may be formed by a combination of supply and demand, time of day, day of the week, and supply and demand alone. One straightforward illustration of this is to raise the price of the product during periods of high demand and lower it during periods of low demand. The model can react instantly to changes in market factors by automatically changing prices in accordance with a predetermined set of rules. The pricing logic in this method is typically quite simple; the main challenge is retrieving underlying factors like supply and demand.

2) Competition pricing: The term "competition-based pricing" refers to a pricing approach in which a company's prices change over time in response to the prices offered by its rivals. This strategy can be used to stay competitive in a market by matching or beating the prices of other companies. The implementation for this model has just one dependency, i.e., access to competition prices.

3) Linear regression: This pricing strategy uses linear regression analysis to ascertain the connection between prices and demand for a product. This can assist in identifying a product's ideal price point in order to maximize sales or profits. The first step is to use historical sales and pricing data to train a linear regression model that can forecast demand based on various prices. Different linear regression methods,

such as simple linear regression, multiple linear regression, and sophisticated techniques, can be used to train the model. The product's demand is then predicted at various price points using the model, which is then used to determine the price that will generate the most revenue (or profits).

Price = b0 + b1.Demand + b2.CompPrice + b3.Time (3)

Eq. (3) shows the outcome of linear regression analysis, is an expression that can be programmed into pricing model to calculate prices over the period of the time, In the equation, b0 represents the baseline price of the product, b1 represents the elasticity of demand, b2 represents the effect of competition on price, and b3 represents the effect of time of day on price.

4) Linear programming: Linear programming (LP) is an optimization method in which the objective function and all constraints are linear. Linear programming dynamic pricing is a pricing strategy that employs linear programming to calculate the best price for a product. The dynamic pricing problem can be expressed as a linear program, with the goal of maximizing revenue or profits while keeping prices, demand, and costs in check. The objective function and constraints are expressed as linear equations, with the solution being the set of prices that maximizes the objective function [19] (Eq. (4)) while satisfying all constraints (Eq. (5) to Eq. (9)).

$$Y_{(revenue)} = \sum_{t=1}^{t=T} \sum_{k=1}^{2} P_t \, Q_{tk} S_{tk} \quad (4)$$

Constraints:

$$\forall k \in K, RC_k \geq \sum Q_{tk} \quad (5)$$

$$T_{(target\ sale)} \geq \sum_{t=1}^{t=T} \sum_{k=1}^{2} Q_{tk} S_{tk} \quad (6)$$

$$1 = \sum_{k=1}^{2} C_k \quad (7)$$

$$P_t := \{ P_t \in \mathbb{R}^n \mid P_{min} \leq P_t \leq P_{max} \} \quad (8)$$

$$Q_{tk} := \{ Q_{tk} \in \mathbb{Z}^{mxn} \mid 0 \leq Q_{tk} \} \quad (9)$$

Where:

R is the total expected number of price requests across k customer segments.

K is the customer segments (K = 2 for myopic and strategic customers setup),

C_k is the ratio of expected request count in the customer category (k) and total requests.

Q_{tk} is number of price response by the DP engine with price P_t in customer segment k. (This is LP Variable).

P_t is the price point to offer, its range is Pmin and Pmax.

S_{tk} is the sale probability at price P_t, customer segment k.

E. Pricing Cache

Client price requests for a product many times more frequent than change in market factors hence the price calculated by model can be cached to avoid the model invocation time, which is not just time but also computationally expensive. Pricing cache decouples the expensive pricing process from servicing the pricing. If cache is not used, price request latency is given by 10, i.e., sum of latencies of model evaluation and sum of all the analytic fetch.

$$Lat_{pr} = Lat_{price-model} + \sum Lat_{analytics} \quad (10)$$

By using the cache, the price-request-latency simply becomes the latency of cache lookup (11).

$$Lat_{pr} = Lat_{cache} \quad (11)$$

Please note that, the pricing cache is considered large enough to hold the calculated pricing results for all the items, and hence cache miss penalties are not considered.

F. Product Module

A product module is a component or module that manages and provides access to product information. This includes information such as the pricing model, minimum and maximum price settings, available inventory, price exploration factor, and any other relevant settings required for the product's pricing model to function. This data is typically stationary and varies infrequently (say, monthly or quarterly). Also note that the product module does not need to exclusive to pricing system; it can be shared across other e-commerce services (such as catalogue, order etc).

VI. AWS Cloud Implementation

We will go over the specifics of this work's cloud implementation in this section. Real-time pricing system implementation requires compute, a database, and the ability to react to events. Most cloud service providers can easily meet these requirements, but we have chosen to implement AWS for the proof-of-concept and because it is the most widely used cloud service. Python was used exclusively throughout the work to program everything.

A. A Quick Review of AWS Services

Amazon Elastic Compute Cloud (EC2) is an AWS web service that allows users to rent virtual computers to run their own computer programs. Elastic Container Service (ECS) is a fully managed container orchestration service that allows you to run and scale containerized applications on a cluster of EC2 instances using Docker and Kubernetes. Amazon Elastic Container Registry (ECR) is a fully managed Docker container registry hosted by Amazon Web Services (AWS). Docker images can be safely and saleably stored, managed, and deployed using ECR. EventBridge is a serverless event bus service that connects applications by automating event scheduling and routing from sources such as DynamoDB to destinations such as Lambda functions, Lambda is a serverless computing service that allows you to run code without the need for server provisioning or management. Lambda-powered applications can respond to events and run functions automatically in response to triggers such as changes to data in DynamoDB or an EventBridge-triggered event. Lambda does have some limitations, such as a maximum amount of memory and limited concurrency, which should be considered for scalability. DynamoDB is a fully managed NoSQL key-value and document database that scales to provide millisecond performance. It is an excellent choice for applications that require consistent, fast performance as well as the ability to

manage large amounts of data. Applications can use DynamoDB Streams to react to changes in DynamoDB tables and invoke AWS Lambda to handle the change [28].

B. Component Implementation

1) Competition watcher: The goal of the Competition Watcher software is to obtain competitor product pricing information by scraping competitor websites. Web-scraping tools are commonly used for this purpose. Several libraries and frameworks, such as "Beautiful Soup," [29] "Scrapy," and "Selenium," are available for web scraping in the Python programming language. Under Amazon Web Services, there are two possibilities for designing and building this:

a) Event-bridge-based periodic Lambda invocation: EventBridge allows you to schedule a Lambda for a specific time (Say every 30 minutes). Lambda executes the web scraping business logic and updates the competition prices in DynamoDB. DynamoDB Stream can then be used to retrigger the pricing process by triggering a lambda function.

b) Docker workflow with ECS, ECR (running on EC2): In this approach, web-scraping logic is executed in a Docker container powered by EC2, ECS, and ECR Services to retrieve competition prices, detect changes, and store in DynamoDB. A DynamoDB stream, like the previous option, can be used to trigger a lambda function for pricing. This process can monitor hundreds of products, making it more cost-effective for large e-commerce setups.

2) Demand watcher: The demand watcher watches the ongoing demand periodically, and if there is a change in the demand, it is supposed to trigger the repricing by invoking the lambda function. Demand Watcher implementation is dependent on E-Commerce architecture implementation. Assume that E-commerce architecture has a database table called REQUSET and a REQUEST_TIME index on it. Demand SQL expression is shown in Fig. 3.

```
SELECT count(*) from REQUEST
where REQUST_TIME >  Now()-POLL_WINDOW
```

Fig. 3. Demand calculation in a period using existing RDBMS DB.

In many use cases, the above approach would work, except when users want more real-time demand tracking. The alternative is to use streaming analytics, for this purpose, first, the incoming price requests can be posted to a Kinesis analytics stream. This stream has a consumer, a lambda function that is configured with batch triggers on Kinesis updates. The lambda analyses and records the demand information in real-time into a DynamoDB. Dynamodb stream then triggers the pricing function to recalculate the price for the new demand (Fig. 4).

3) Customer segmentation and sale probability: For the analytics that do not change in real-time, like sale-probability and customer-segments, in order to be cost-effective, the framework would compute these periodically (say, once a day). We use the price request stream and order stream, and register lambda triggers (with the largest possible batch size/window, i.e., 5K records and 5 minutes). Both the lambdas add/update the entries in the Analytics table (Fig. 5). Their logic updates and aggregates the incoming stream in the format shown in Table I. This serves as the common source for segmentation and sale probability calculators. Customer segmentation is done by aggregating the analytics table by customer-id and the ratio of purchases to requests. Those with a higher purchase to request ratio are classified as myopic, while the rest are classified as strategic. Table II shows DynamoDB schema for customer segmentation table. Sale-probability is calculated as sale to request ratio for each segment and at each price points. Table III displays the schema for DynamoDB table for the same. Please Note, since DynamoDB is a KV store, probability data is stored in json format, and pricing-model is expected to deserialize that before using.

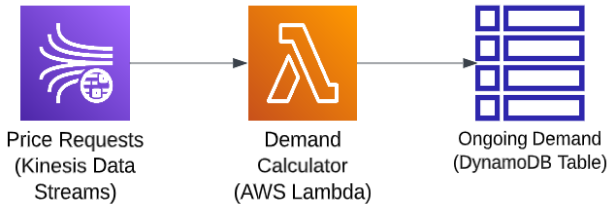

Fig. 4. Realtime demand-data accumulation.

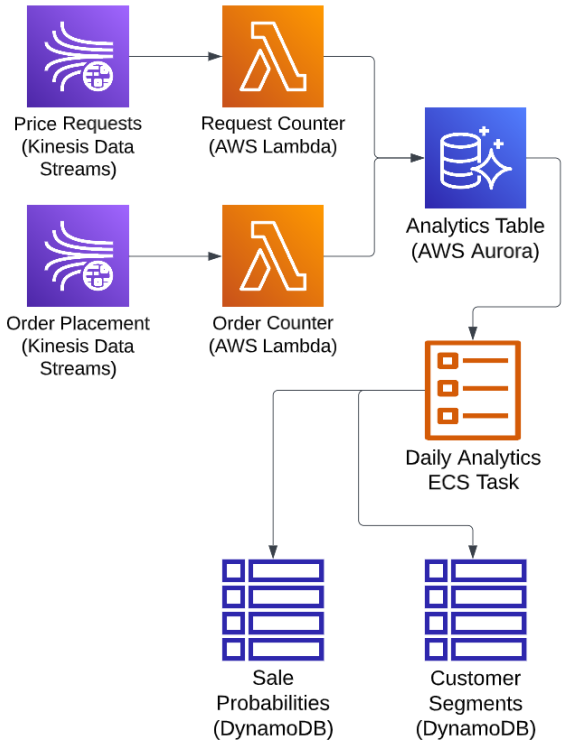

Fig. 5. Analytics: sale probability & customer segmentation.

TABLE I. ANALYTICS TABLE SCHEMA

Date	String	String	Float	Bool
Date	Cust-id	Product	Price	Purchased

TABLE II. CUSTOMER SEGMENTATION SCHEMA

String	Integer
Cust-id (key)	Segment

TABLE III. SALE PROBABILITY SCHEMA

String	String (Json)
Product (Key)	{segment_str = [[price_float, probability_float]] }

4) Pricing model and price-cache: The framework proposed in this work supports multiple pricing methods; all of those are implemented in python and deployed with AWS Lambda. Below picture shows the interaction of Pricing logic with its upstream data-sources. The pricing logic may be invoked in real-time in response to changes in Demand and Competition Prices (depending on the pricing strategy).

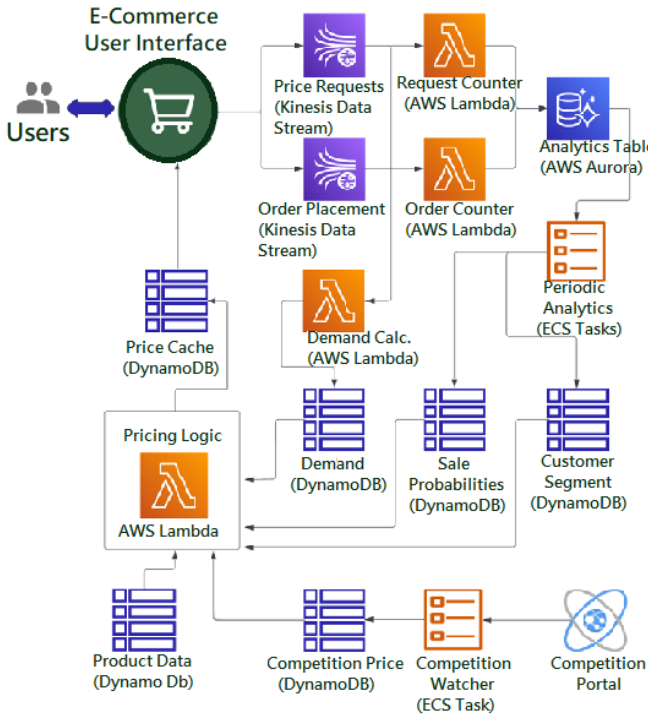

Fig. 6. AWS architecture for real-time pricing.

Additionally, if pricing strategy depends on customer segmentation and/or sales probability, the same are fetched from the respective Dynamodb tables. Pricing logic writes its data to Price Cache. As stated before, Pricing clients connect to pricing cache and receive real-time dynamic pricing with latency as low as a key lookup into database.

C. Complete AWS Architecture

Fig. 6 presents the overall end-to-end system architecture; it glues all the components described in the previous subsection and demonstrates their connection and dependencies. The figure describes the cloud-based application as a set of connected microservices that communicate with each other over the network. This design is quite modular (with clear separation of concerns) and flexible, which makes it easy to substitute individual components. In the previous section, multiple pricing models were described. It is entirely possible to replace an existing pricing model with a new and better one without changing other parts of the system. To be specific in the context of AWS, it is all about redefining the lambda expression for the pricing function, which follows the same interface (I/O contract).

VII. RESULTS

A. Experimental Setup

Our experimental setup focuses on both functional and non-functional requirements. For functional requirements such as the behavior of the pricing model, the implementation uses Python-based simulation for pricing logic. The non-functional requirement was verified by setting up the component, as shown in Fig. 6, in our test AWS account.

B. Experimental Result

Fig. 7, 8, and 9 show how competition-based pricing works. Because the goal of such a pricing mechanism is to simply follow market pricing, a re-pricing signal is easy to find. We implement a module that periodically scrapes prices from popular e-commerce websites. The latency of scraping, for the competition watcher module, is shown in Fig. 7, which was found to be around 600-800 ms. Please note that this latency also includes network latency.

Fig. 8 and Fig. 9 show the tracking of competition prices. Fig. 9 is a zoomed in version of Fig. 8 to show the difference in timing between the price signal and price change.

Though it takes a while for market-watcher component to observe the competition's price changes, however, thanks to DynamoDB trigger, lambda evaluation, and subsequent price-cache update is quite fast (double digit of milliseconds). The whole reaction time to competition price is around one second, which should be acceptable reaction latency in almost all real-time pricing use-case.

Fig. 10 shows a simulation of rule-based dynamic pricing. The rule is a combination of demand for the product and the competition's price, with an upper cap on the maximum price. The rule provides separate prices for myopic and strategic customers.

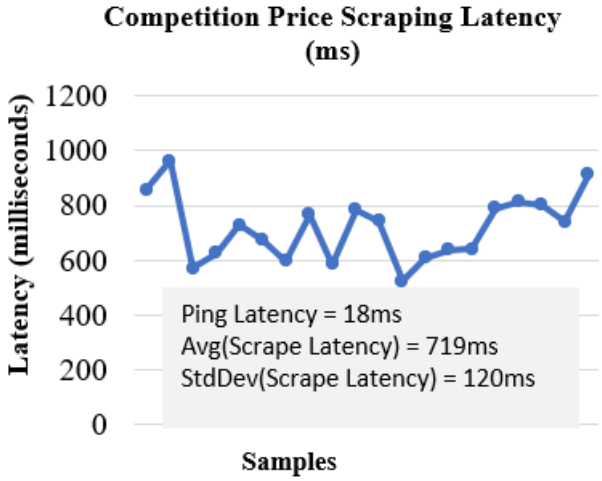

Fig. 7. Competition price fetch latency.

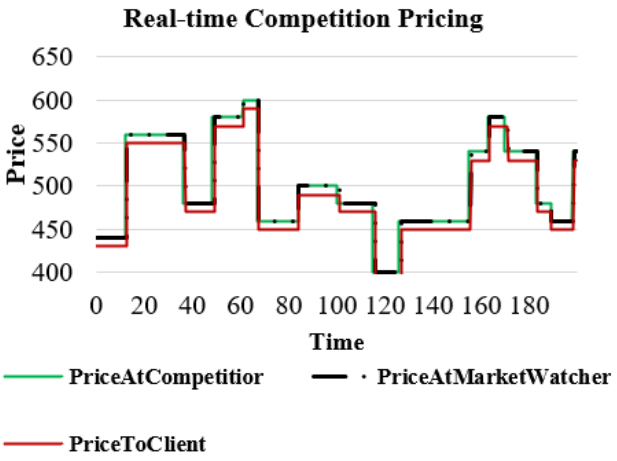

Fig. 8. Competition pricing tracking.

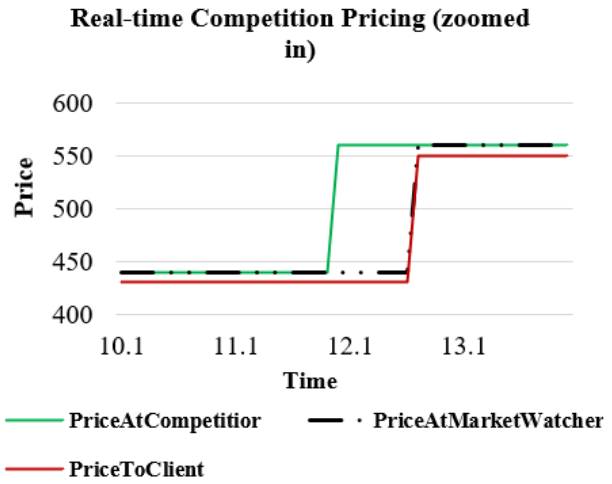

Fig. 9. Competition response latency.

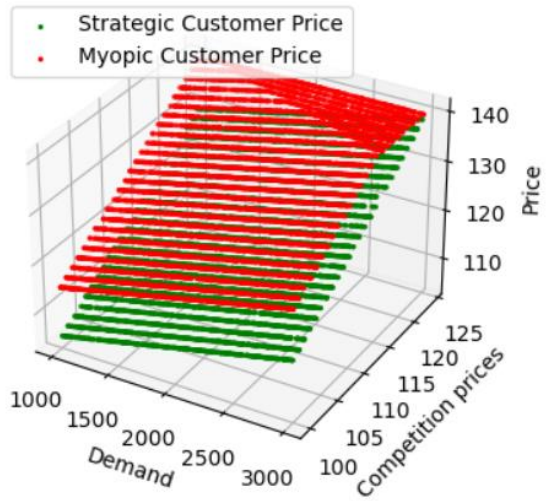

Fig. 10. Rule-based dynamic pricing.

Fig. 11 and 12 are the simulation outcomes for the linear programming-based pricing model. The solution for objective function (Eq. (4)) and constraint (Eq. (5) to Eq. (9)) can change the prices for changing demand or target sale-quantities and maximize the revenue under the different circumstances. Fig. 11 shows optimal prices for different values of target sale-quantity, and Fig. 12 shows the same different demand values.

Fig. 13 and Fig. 14 explain the content and performance of the price cache. Fig. 13 is a snippet taken from AWS console depicting the content of price cache. Key shown in the Fig. 13 is the product-id, and the json value provides the price values for myopic and strategic customer.

It also has a default value, which is useful in situation when pricing-model is yet to provide any pricing value. Fig. 14 shows our typical distribution of price-request latency when using DynamoDB as a pricing cache. The latency is single-digit millisecond latency and comparable to static-pricing scenarios.

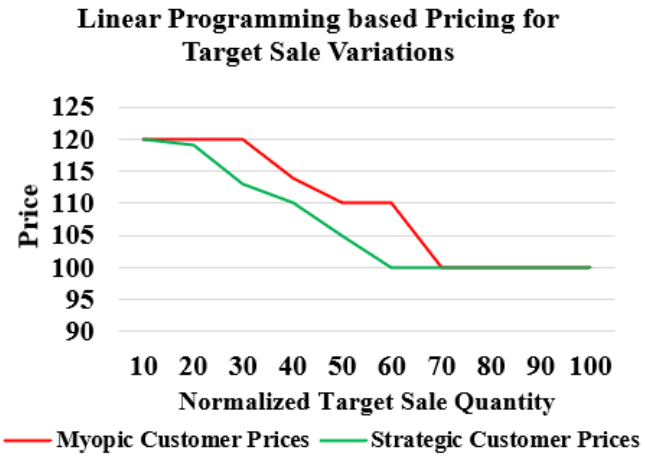

Fig. 11. Linear programming-based pricing (1).

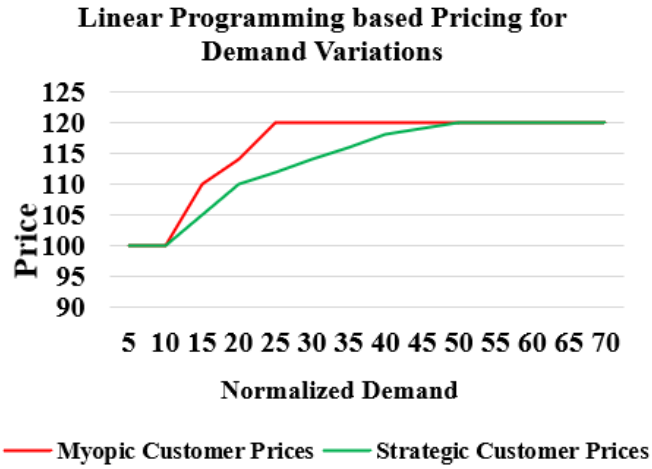

Fig. 12. Linear programming-based pricing (2).

	key1	json
☐	np374	{"default_price": 3500, "prices": [[[1, 3840]], [[1, 3835]]]}
☐	np623	{"default_price": 7000, "prices": [[[1, 6330]], [[1, 6325]]]}

Fig. 13. Typical price cache values.

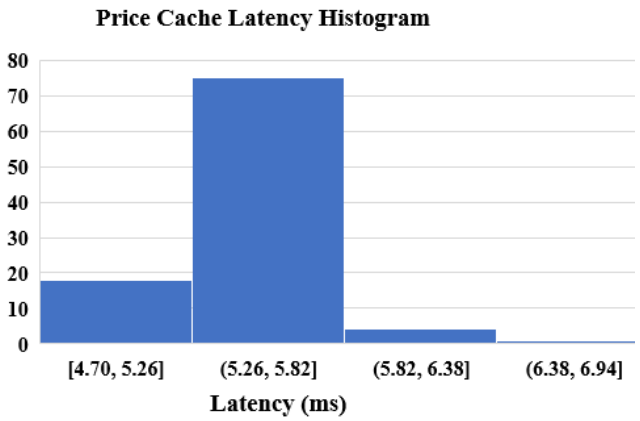

Fig. 14. Price cache latency distribution.

VIII. Conclusion

A. Summary and Discussion

Real-time pricing can help e-commerce owners; however, implementation requires careful handling of multiple requirements in order to perform dynamic pricing in real-time. Through this work, we defined functional and non-functional requirements and used them to guide our system design process. We created a pricing framework that can potentially provide the most suitable price for various products at different stages of their product lifecycle while taking a variety of market factors such as demand and competition into consideration (satisfying functional requirements FR1 and FR2).

The components (e.g., demand watcher, competition price fetcher, cache) are reusable, concurrent (designed with separation of concerns in mind), and allow plug-and-play of different pricing logic through the defined database schema (satisfies NFR1). By the virtue of design decision, price response latency (NFR2) equals to a cache-lookup which is similar to systems without dynamic pricing. The cache latency can further be improved (if required) by using DynamoDB-DAX [30]. It has been demonstrated in the result that, system response to any price change is within a single second, an acceptable latency in the context of e-commerce (NFR3). The cloud native building blocks (DynamoDB, lambda, etc.) inherently provide availability and scalability to our design and with the pay-as-you-go model for cost, cloud-native design satisfies the last one, NFR-4.

B. Research Contributions

Real-time e-commerce pricing is a relatively unexplored academic field. In the research gap analysis, we did not find any implementation-specific work, which emphasizes the timeliness of the pricing. As our first research contribution, we have been able to apply existing knowledge of 'dynamic pricing' to methodologically specify, design, and implement a real-time pricing system.

Other contribution of this research is the demonstration of cloud-native design principles for the implementation of real-time pricing. The framework provides, unlike anything else in the published literature, a low-cost (pay as you go) and scalable solution for real-time pricing by making use of cloud implementation methodologies. By providing the ability to plug-and-play a pricing model for different e-commerce use-cases, the framework does not enforce 'one-size-fits-all', but rather promises flexibility while maintaining the reusability of building blocks.

C. Future Work

The primary goal of this work is to create a framework for a real-time pricing system, which has been demonstrated in the implementation and results section. However, because real-time pricing for e-commerce is going to evolve further, there are a few opportunities for future work:

- There is an opportunity to improve the analytics described by performing some finer optimizations. To give an example, advanced statistical techniques can be used to provide more accurate results for sale-probability estimation. Similarly, sophisticated forecasting techniques can be utilized to predict demand, which will help price the product more efficiently over the long run.
- Another potential area of research can be conducted on pricing models and their selection in the context of real-time pricing and cloud native. Models can be fast or time-consuming and may have different memory and compute requirements, leading to deployment choices such as serverless or containers.

References

[1] Y. Narahari, C. Raju, K. Ravikumar, and S. Shah, "Dynamic pricing models for electronic business," *Sadhana*, vol. 30, pp. 231–256, 2005.

[2] D. Elreedy, A. F. Atiya and S. I. Shaheen, "Multi-Step Look-Ahead Optimization Methods for Dynamic Pricing With Demand Learning," in *IEEE Access*, vol. 9, pp. 88478-88497, 2021, doi: 10.1109/ACCESS.2021.3087577.

[3] S. Christ, *Operationalizing Dynamic Pricing Models: Bayesian Demand Forecasting and Customer Choice Modeling for Low Cost Carriers*. 2011.

[4] G. Mehra, "Real-time Pricing Affordable for Smaller Merchants," Practical Ecommerce, Jan. 22, 2017.https://www.practicalecommerce.com/Real-time-Pricing-Affordable-for-Smaller-Merchants.

[5] "IDC Forecasts Worldwide," IDC: The premier global market intelligence company.https://www.idc.com/getdoc.jsp?containerId=prUS48208321.

[6] Z. Mahmood, "Cloud computing for enterprise architectures: concepts, principles and approaches," in *Cloud computing for Enterprise architectures*, Springer, 2011, pp. 3–19.

[7] M. Wu, X. Ding, and R. Hou, "Design and implementation of B2B E-commerce platform based on microservices architecture," in *Proceedings of the 2nd International Conference on Computer Science and Software Engineering*, 2019, pp. 30–34.

[8] S. Athreya, S. Kurian, A. Dange, and S. Bhatsangave, "Implementation of Serverless E-Commerce Mobile Application," in *2022 2nd International Conference on Intelligent Technologies (CONIT)*, 2022, pp. 1–5.

[9] "Cloud Computing Services - Amazon Web Services (AWS)," *Amazon Web Services, Inc.* https://aws.amazon.com/.

[10] "Empower Your Business in USA & Canada with Alibaba Cloud's Cloud Products & Services," *Empower Your Business in USA & Canada with Alibaba Cloud's Cloud Products & Services*. https://www.alibabacloud.com.

[11] "Start and grow your e-commerce business - 3-Day Free Trial," *Start and grow your e-commerce business - 3-Day Free Trial*. https://www.shopify.com/?ref=mile-high-themes.

[12] "What Is Magento Ecommerce And Why Should You Use It?," *World's #1 POS for Magento*, Mar. 04, 2021. https://www.magestore.com/blog/what-is-magento/.

[13] I. Yeoman, "The history of revenue and pricing management – 15 years and more," Journal of Revenue and Pricing Management, vol. 15, no. 3–4, pp. 185–196, Jun. 2016, doi: 10.1057/rpm.2016.36.

[14] Kokkoris, I., & Lemus, C. (2022). *Research Handbook on the Law and Economics of Competition Enforcement*. E-CONTENT GENERIC VENDOR. https://books.google.co.in/books?id=jqKCEAAAQBAJ.

[15] T. Wang *et al.*, "A framework for airfare price prediction: a machine learning approach," in *2019 IEEE 20th international conference on information reuse and integration for data science (IRI)*, 2019, pp. 200–207.

[16] Gallego, G., & Topaloglu, H. (2019). *Revenue Management and Pricing Analytics*. Springer New York. https://books.google.co.in/books?id=YF-pDwAAQBAJ.

[17] S. Saharan, S. Bawa, and N. Kumar, "Dynamic pricing techniques for Intelligent Transportation System in smart cities: A systematic review," *Computer Communications*, vol. 150, pp. 603–625, 2020.

[18] R. Phillips, *Pricing and Revenue Optimization*. Stanford University Press, 2005. [Online]. Available: https://books.google.co.in/books?id=Xi17Xx9rD9wC.

[19] A. Kumari and B. R. K, "Design of a Real-Time Pricing System for E-commerce," *International Journal of Computer Theory and Engineering*, vol. 15, no. 1, pp. 46–53, 2023.

[20] S. Kedia, S. Jain, and A. Sharma, "Price optimization in fashion e-commerce," *arXiv preprint arXiv:2007.05216*, 2020.

[21] A. V. Den Boer, "Dynamic pricing and learning: historical origins, current research, and new directions," *Surveys in operations research and management science*, vol. 20, no. 1, pp. 1–18, 2015.

[22] A. X. Carvalho and M. L. Puterman, "Dynamic pricing and reinforcement learning," in *Proceedings of the International Joint Conference on Neural Networks, 2003.*, 2003, vol. 4, pp. 2916–2921.

[23] D. Gannon, R. Barga, and N. Sundaresan, "Cloud-native applications," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 16–21, 2017.

[24] J. Lee and Y. Kim, "A Design of MANO System for Cloud Native Infrastructure," *2021 International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju Island, Korea, Republic of, 2021, pp. 1336-1339, doi: 10.1109/ICTC52510.2021.9620858.

[25] G. -C. Pan, P. Liu and J. -J. Wu, "A Cloud-Native Online Judge System," *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*, Los Alamitos, CA, USA, 2022, pp. 1293-1298, doi: 10.1109/COMPSAC54236.2022.00204.

[26] H. Lang, H. Tian, D. Li, Z. Niu and L. Wen, "Design of A Cloud Native-Based Integrated Management Platform for Smart Operation of Multi-Business Buildings," *2022 14th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, Hangzhou, China, 2022, pp. 169-173, doi: 10.1109/IHMSC55436.2022.00047.

[27] P. Offermann, O. Levina, M. Schönherr, and U. Bub, "Outline of a Design Science Research Process," In *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology* (pp. 1-11). 2009. doi: 10.1145/1555619.1555629.

[28] Cloud Products," *Amazon Web Services, Inc.* https://aws.amazon.com/products/.

[29] L. Richardson, "Beautiful Soup: We called him Tortoise because he taught us.," *Beautiful Soup: We called him Tortoise because he taught us*. https://www.crummy.com/software/BeautifulSoup/.

[30] "In-memory acceleration with DynamoDB Accelerator (DAX) - Amazon DynamoDB," *In-memory acceleration with DynamoDB Accelerator (DAX) - Amazon DynamoDB*. https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DAX.html.