Texas A&M University

Mays Business School

Department of Information and Operations Management

ISYS 622 Advanced Data Management

Elastic Pricing of Books

Section 603, Group 10:

Aditya Purandare

Swapnil Phulse

Vijaylakshmi Rana

Vikramsinh Jadhav

Guide:

Dr. Matthew Manley

December 6, 2015

Contents

1. Introduction

The advent of 21st century has ushered in usage of mobile devices, personal computers and PDAs. This has resulted in drastic reduction of usage of printed books. Moreover standard pricing of these books that stays put at the same value throughout its lifetime deters many potential users from buying. Figure below rightfully shows how sales for printed book has nd will reduce over the course of time.

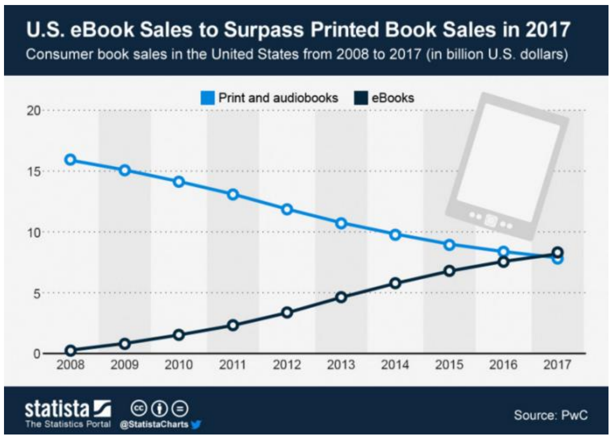


Figure 1Price estimates for books

Revenue from book sales generated on ecommerce websites has seen a decline as books age over a period of time. Newer editions being published, new books have been released on the same domain resulting in stagnation of sales revenue for a book over its lifetime. We introduce an elastic pricing strategy for books & eBooks which leads to increased revenue generated over the life span of book. In terms of data storage, MongoDB - a document-based NoSQL Database can scale to store/retrieve 1,000,000 books in the catalogue and to track millions of dollars worth of book sales throughout a month. Apache Hadoop will be designed to compute book prices based on its historical sales and predictive metrics, which will then match book prices to periodic sales to bring in more revenue for ecommerce Platform (Case Study: Amazon Books) and also for the book publishers & authors.

1.1. Problem Statement

We believe that an effective pricing strategy can resulted in increased sales of the book, canvassing various factors like seasonality of demand and its depreciation to come up with an ideal predicted price. Our research has led to creation of a new mathematical model that can calculate this optimum price resulting in increase in sales by 5% to 21% (data specific to books used during this analysis).

1.2. Technology stack and other tools

After working with various tools and languages, we have zeroed in on using the following:

MongoDB for database storage

When it was time to choose our production databases amongst our design options namely Cassandra, HBase CouchDB and MongoDB, our business requirements and initial test bed results on trying these out on a small scale increased our inclination towards MongoDB. MongoDB excels when applied to large datasets that require queries to span the entire corpus. MongoDB's sharding functionality can be leveraged to handle this class of problem. We believe it was efficiently used to perform queries on our single monolithic dataset. MongoChef has been used as supporting client for the same, while MongoLab is being used as online server to host the database.

Apache Hadoop for MapReduce

Benchmarking two options viz Apache Spark + MongoDB and Apache Hadoop +MongoDB as computation platforms, we found that Spark has an edge over Hadoop when it comes to speed. Spark handles most of its operations “in memory” – copying them from the distributed physical storage into far faster logical RAM memory. But Hadoop performance is sufficient enough and it led to high developer productivity owing to our class assignments submitted throughout the course.

JavaScript, jquery, HTML and css as frontend for output visualization

The stack combination above has been used to build required web pages that serve as dashboards to view online output. Seamless integration with both HDFS and MongoDB drove this decision.

GitHub for version control

Building this application required every team-member to contribute to our final codebase. After learning from initial code revisions when multiple people are collaborating on a project, it's really hard to keep track of who changed what, and to keep track of the revisions that took place. GitHub takes care of this problem by keeping track of all the changes that have been pushed to the repository.

Trello as Project Management tool

Coordinating this project was made easy on account of using Trello. It lets us create Activities, Assignments to members, Milestones and Deliverable schedules, Responsibility Cards etc. Its heavy use helped us track dependencies and risks involved throughout the project.

Azure as deployment service

Our research and project has been appreciated by a Product Manager from Microsoft during Hackathon and won us a premium Microsoft Azure subscription for three complete years. It will be used to host final product online.

1.3. Roles and Responsibilities

Project Role	**Name**	**Project Responsibilities**
Guide	Dr Mathew Manley, Rakesh Kaushik	• Approve project initiation plan • Approve any changes to the plan, scope or timeline • Receive and review project status reports
Strategy design expert	Vijaylakshmi Rana, Vikramsinh Jadhav	• Champion the project • Design implementation plan, project scope and milestones • Resolve strategic issues • Prioritize project goals
System expert	Aditya Purandare, Swapnil Phulse	• Participate actively in functional specification development meetings • Write / assist with functional specification document(s) • Revisit and prioritize requirement specifications, functional specifications
System programmer	Aditya Purandare, Swapnil Phulse, Vijaylakshmi Rana	• Develop system interfaces (read Data Access Layer) • Design, build, integrate, unit and system test, create all technical deliverables • Test system interface(s)
Data Requirement Expert	Aditya Purandare, Swapnil Phulse, Vikramsinh Jadhav	• Identify data requirements based on business needs • Identify data options and resources
End User Documentation	Vijaylakshmi Rana, Vikramsinh Jadhav	• Participate in testing • Develop documentation

2. Analysis and Design

2.1. Business Requirements

With pricing elasticity, we intend to solve the problem of determining optimum price range in conjunction with maintaining a constant incoming revenue stream for a book. This Proof-of-Concept can be leveraged in various applications that model quantitative response to increasing age and lowering demand of any commodity. Striking a golden price range that boosts sales revenue can be beneficial to various parties:

a) Customers pay less
b) Authors get royalty checks of larger amounts that correspond to the larger audiences
c) Even though individual customers pay less, publishers and retailers receive higher share of total revenue generated

At the onset, we have identified two perspectives that define price elasticity for a book:

a) Own price elasticity – Determining change in price for one single book sale corresponding to its demand
b) Cross price elasticity – Determining changes in price for one single book due to changes in price and demand of another book. This stays out of scope in our current application.

2.2. Analysis

2.2.1. Price Elasticity

Before defining price elasticity in terms of books, let us first see what elasticity means as follows

It is a measure of responsiveness. The responsiveness of behavior measured by variable Z to a change in environment variable Y is the change in Z observed in response to a change in Y. Specifically, this approximation is common:

elasticity = (percentage change in Z) / (percentage change in Y)

The smaller the percentage change in Y is practical, the better the measure is and the closer it is to the intended theoretically perfect measure.

In our case, we calculate price elasticity of demand. So it denotes a change in quantity (demand) of books resulting due to change in book price. This inverse dependency and seasonal nature of book sales will be used going forward to estimate prices needed.

$$\eta = \frac{\frac{Qstart - Qend}{Qstart + qend}}{\frac{Pstart - Pend}{Pstart + Pend}}$$

Where

Qstart = Number of books sold during datamonth n

Qend = Number of books sold during datamonth n+1

Pstart = Average price of book during datamonth n

Pend = Average price of book during datamonth n+1

2.2.2. Depreciation

Depreciation defines the reduced value of any asset over its lifetime. Price calculated after deducting depreciation i.e. its book value will be called '*Depreciation Factor*'. We have formulated it as follows:

$$Depreciation factor = Poriginal \times (1 - (\frac{Ycurrent - Yoriginal}{Age}))$$

Where

Depreciation factor = Price of book after depreciation

Poriginal = Original publishing price of the book

Ycurrent = Year of analysis

Yoriginal = Year when book was published first

2.2.3. Proposed Mathematical Modeling

Prices vary frequently based on several factors like present economic conditions, seasonality, competitive publishers and use-value for it. Bookscan, Nielsen and Bowker all provide general numbers and stats with the limited numbers they track, and the sales numbers can be vastly different.

We have created a mathematical model that brings two different price factors(namely Forecast price factor +Depreciation price factor) and outputs an ideal price. The formulae to calculate it given below as:

$$Pforecasted = (1 + GR) * \frac{\eta}{\eta + 1} * [\frac{Pactual}{2} * IsCurrent + \frac{Ppredicted}{2} * (1 - IsCurrent)$$

Where

GR = Growth rate for book defined in input file

η = Price elasticity for book

Pactual = Actual price as present in baseline data

Ppredicted =Forecasted price for book

IsCurrent = Flag indicator for current year. Its value is 1 for current year and 0 for past/forecasted data

Idea price (Pideal) will be calculated using depreciation factor calculated above and Pforecasted mentioned above as follows:

$$Pideal = w1 * Pforecasted + w2 * Pdepreciated$$

Where

w1 and w2 denote weights that are calculated in Reducer 4.

The best pair of weights is calculated using historical values.

2.3. Data Requirements

Data requirements imply actual data in a particular format required for the application to function as specified. As the technology advances, volume of data from various sources increases at very high rate which makes it difficult to handle, process, and store the data. Another problem with fast pace of technology advancement is frequently changing data formats and additional data attributes we get, poses challenge of modifying the existing structure of stored data. The application is going to get data from Google Books API which provides book information, and eBook availability. The book pricing application needs data about the sales of books, publication year, and original price of book at the time of publication.

The other data required is total quantity sold for each book which can be used to calculate the price elasticity of demand (η). The application also requires data for growth rate that is to be attained for every book.

We also need the following data characteristics of the data attributes

- Data element name
- Synonymous name
- DataType (e.g., alpha, alphanumeric, or numeric)
- Definition
- Format
- Range of values or discrete values
- Unit of measurement
- Precision (e.g., number of decimal places)
- Data item names, abbreviations, and codes
- Characteristics, such as accuracy, validity, timing, and capacity

Data Constraints

The main purpose of the application is to address two main issues of big data. First is large volume of data processing and storing it in retrievable formats. Second is creating a structure so that it can be modified easily in the future. The combination of MongoDB which is a NoSQL database and Apache Hadoop resolves these two issues. The data base is scalable in terms of data storage and processing. Main constraint on MongoDB is it can perform operations or transaction on only one collection so it's not possible to join the data from two separate collections.

Data Retention

Data retention requirements for the book pricing application is mostly based on idea of future prediction of prices based on multiple factors. Book price data for current year and previous year is stored into the database. This data is stored in a format such that it caters to need of input data for mapper which is then used to calculate ideal prices of the books. The forecast data for future 1 year is stored in to the text file which serves as an input file to website front end for output visualization.

Measurement Conversion Factors

Sales data of books for last two years is stored in the database. For projecting future sales companies set their own growth rates based on organizational strategy. Future sales conversion is done based on this growth rate.

Frequency of Update and Processing

The expected frequency of data element change is on daily basis as everyday books sold and prices will change. This data can be pulled and stored into MongoDB using scheduled batch scripts that run regularly.

2.3.1. Data Quality

The data quality is always extremely important aspect of web applications. Poor quality of data impacts business adversely results in unfavorable situations. Hence for an organization, crucial data management strategy is ensuring data accuracy, consistency and completeness. The data quality of input has been verified by analyzing it through various tests like null value test for BookID attribute. Data quality assurance included investigation of missing fields, empty spaces, wrong format, and misaligned data to improve overall understandability, use, and trust about the data. This activity has significant role to play in supporting our implementation.

Data Profiling

Data profiling is analyzing the input data in a specific way to discover and characterize important features of data sets. Input data selected from Google Book API has Json format which includes lots of attributes. This data is analyzed to find out useful attributes like book price, quantity of books sold, price of the book which are required to find out ideal prices. Output of data profiling activity produced, data structure, content, rules and relationships granularity, value sets, format patterns, content patterns, and implied rules among the various data attributes. This analysis helped to identify data anomalies, determine their potential business impact and develop dimensions for measuring data quality level.

Error Handling

Error handling processes are adopted to identify inaccurate or incomplete data. If the input data format is not as expected then application will exit with error saying input data format is not correct. If the book price after depreciation is less than the half of original publishes price then application will take the depreciated price as half of the original publish price. Java program handles many other exceptions and errors which can be caused by erroneous data or otherwise.

Metadata management

Metadata is data about data. It is vital to maintain a shared library of all the attributes and definitions, glossaries of business terms, definitions of data elements and internal standards on data architecture, data modeling, naming conventions and data exchange methodologies. This application maintains metadata library which users, database admins, and application developers can be use to understand data as well as modify data structures seamlessly across the enterprise.

Data Accuracy

Accuracy of the input data has been maintained as application uses Google Books API. Since Google is globally trusted organization with high standards of data accuracy, the input for application is mostly accurate.

Data Consistency

Data consistency is maintained across the database, input file to MapReduce program, output file from MapReduce program, input file to user interface and the actual displayed values on the webpage. The data has been tested for consistency at every stage of transition.

Data Completeness

Completeness of data is crucial aspect of data quality. The book pricing application involves financial element in it hence it is very sensitive to completeness of data. Data completeness has been tested at the various levels as missing data might cause irregular or inaccurate book prices.

Data Integrity

Data integrity refers to maintaining and assuring the accuracy and consistency of data over its entire life-cycle. The book pricing application involves rigorous testing of the data so as to maintain data integrity. This will help the users trust application data displayed on the interface.

Timeliness

The data timeliness has been taken care by integrating application with the Google Books API data. When user queries the data for particular book then program fetches the data from API and then processes it through the MapReduce framework. Then output is displayed on the user interface. The prime concern here is that if API is not working then application will fail. To avoid this we may show old data which is present in the MongoDB database while API is not working. This functionality can be implemented in future.

2.3.2. Data Security Concerns

Since the origination of NoSQL databases, important requirements such as data security have not been fully addressed. There are three basic requirements for databases management systems - confidentiality, integrity and availability. The stored data must be available when it is needed (availability), but only to authorized entities (confidentiality), and only modified by authorized entities (integrity).

Data files in MongoDB are never encrypted, and there is no method provided to accomplish this. If encryption is needed, the application layer should perform the data encryption before writing to database. Strong file system security is also recommended.

MongoDB is far behind in implementing the desired security logging and monitoring. Most monitoring and reporting tools currently distributed with MongoDB are related to database performance, mainly for showing the running state of a MongoDB instance. If security is not enabled for the MongoDB instance, which is by default, no authorization is needed to access this interface, resulting in a potential vulnerability. When it comes to Data consistency there is a possibility of inherent data inconsistency among clustering nodes.

Therefore, the conclusion that can be drawn based on the above points is that while working with NoSQL databases, we as developers had much greater responsibility in ensuring reliable transactions and data consistency.

2.4. Reassessment and Prioritization

As the project progressed, several risks were identified and functionality was reassessed to confirm with its scope. In pursuit of useful data that could help our analysis, team visited multiple data repository resources. A dataset was finalized based on aggregate design and is being used to perform elastic price estimations

2.5. Use Cases

System performs various tasks that are executed and each of these has a specific operation assigned to it. Since user interaction in this system is minimal, use cases are closely tied to mathematical computations.

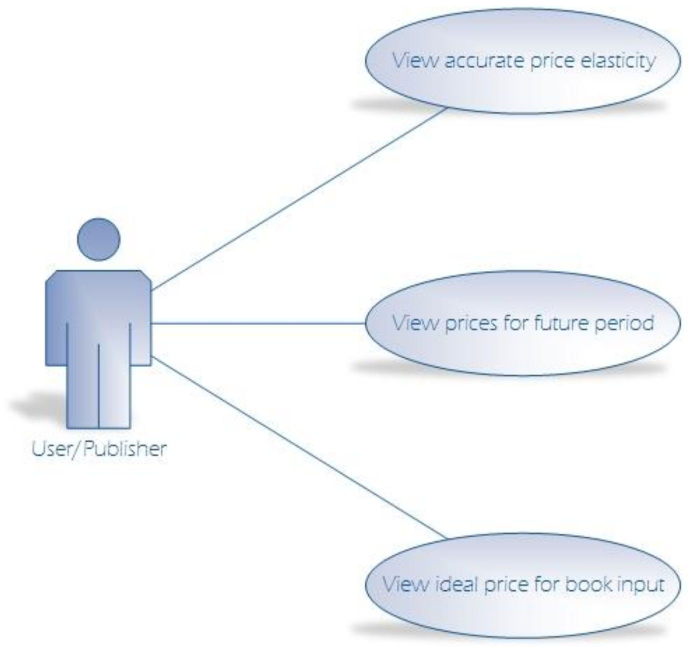

View accurate price elasticity
View prices for future period
View ideal price for book input
User/Publisher

Figure 2 Use Case Diagram

Use case description

a) View accurate price elasticity

Description: As mentioned previously, price elasticity of demand varies based on quantity. The system should calculate price elasticity of demand for each datamonth (yearmonth) so that the user is able to view this price elasticity value for a particular book chosen.

Actor: Publisher

Preconditions: Our system is provided with prices and quantities for various books over the course of 2 years.

b) View prices for future period

Description: The system should calculate future prices so that the user is able to view these future prices for a particular book chosen.

Actor: Publisher

Preconditions: Baseline data used to trend prices should be passed through forecasting algorithm.

c) View ideal price for book input

Description: Fidelity of our system depends on book price calculated using two aforementioned factors. Sales revenues forecasted using these ideal prices will be used by executives to make decisions based on future inventory, marketing schemes, focused sales and ad campaign for books.

Actor: Publisher

Preconditions: Our system is provided with prices and quantities for various books over the course of 2 years.

2.6. Architecture

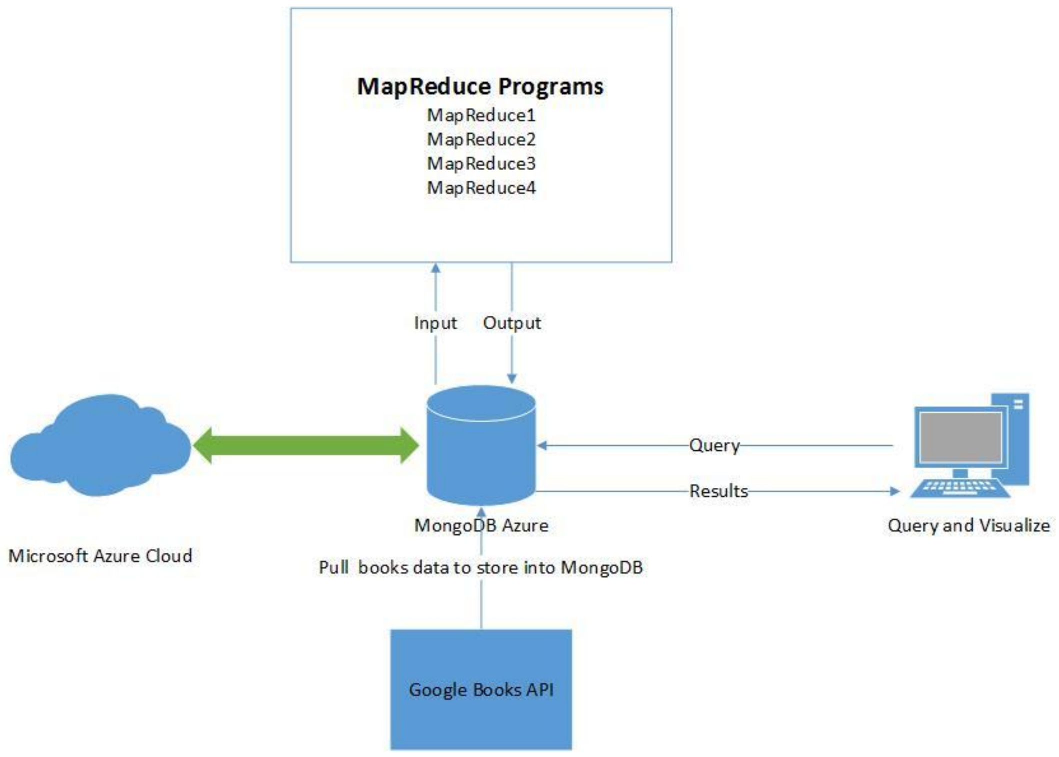


Figure 3 Architecture Diagram

Figure above shows a high level architecture diagram for the project. As seen, data from Google Books API was read, run through an excel template and loaded in MongoDB using MongoChef client for communication. Hadoop MapReduce framework will serve for computing values according to mathematical modeling. These mappers and reducers will be chained together to run without any manual intervention. The outputs will be displayed using a live webpage hosted on Azure cloud. Azure cloud provides best Big Data analysis and Machine learning framework that can easily be used as needed.

2.7. Workflow Diagram

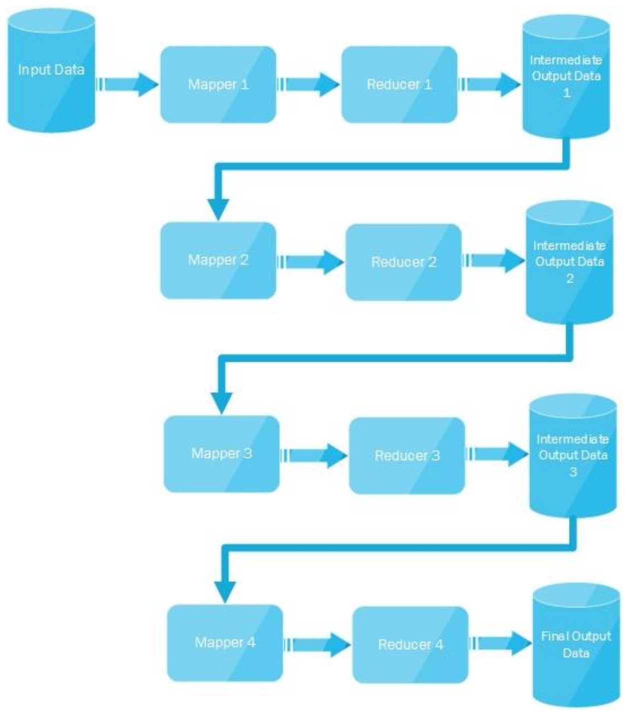


Figure 4 Workflow Diagram

Book sales data as available in aggregate form is converted to a txt file by parsing each of the documents using code in MongoToText.java. This txt file created from MongoDB documents will then serve as an input to Mapper 1. Job chaining is done in driver program (Program.java) to orchestrate flow of key-Value pairs as passed from one task to another.

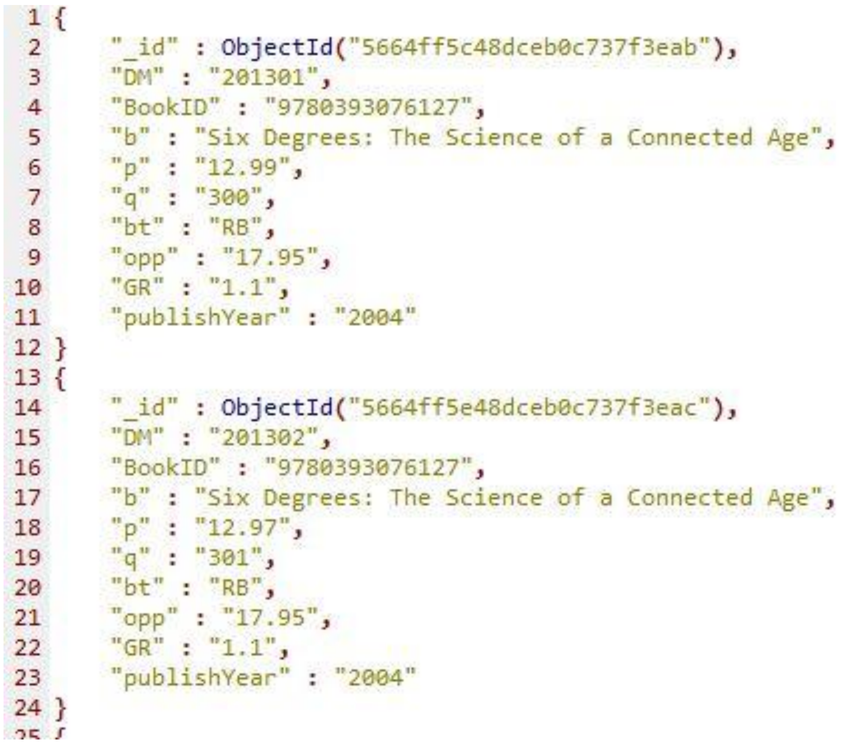
First MapReduce task (comprising of Mapper 1 and Reducer 1) is responsible to calculate price elasticity of demand using formulae mentions in Mathematical modeling (for η). As input is present for books for a baseline period (2 years), reducer will have emitted price elasticity for books for each of these datamonths. Following this, MapReduce task (comprising of Mapper 2 and Reducer 2) will forecast prices for every books for next 12 datamonths to be trended based on baseline prices provided. An exponential smoothing model has been coded in Forecast_Prediction.java which is available.

After that, two factors accounting in price estimation are calculated in MapReduce task (comprising of Mapper 3 and Reducer 3). Section on Mathematical modeling contains all the formulae required to calculate these values which are then used by another MapReduce task (comprising of Mapper 4 and Reducer 4) that evaluates ideal price after applying weight to each of the price factors mentioned before.

3. Implementation

3.1. MongoDB Aggregate Structure

Figure below shows sample aggregate (and its design). As seen, it has been designed such that all information required in processing it through MapReduce is made available through these individual fields like (GR, DM,BookID etc). Please refer document data dictionary to view details.



```
{
    "_id" : ObjectId("5664ff5c48dceb0c737f3eab"),
    "DM" : "201301",
    "BookID" : "9780393076127",
    "b" : "Six Degrees: The Science of a Connected Age",
    "p" : "12.99",
    "q" : "300",
    "bt" : "RB",
    "opp" : "17.95",
    "GR" : "1.1",
    "publishYear" : "2004"
}
{
    "_id" : ObjectId("5664ff5e48dceb0c737f3eac"),
    "DM" : "201302",
    "BookID" : "9780393076127",
    "b" : "Six Degrees: The Science of a Connected Age",
    "p" : "12.97",
    "q" : "301",
    "bt" : "RB",
    "opp" : "17.95",
    "GR" : "1.1",
    "publishYear" : "2004"
}
```

3.1.1. Document Data Definition

Field	**Description**	**Datatype**
_id	This field is used to store the document's ID.	Object ID
DM	This field is used to store the Datamonth. Datamonth is the combination of year and month	String
BookID	This field is used to store the BookID. BookID in this case is the ISBN.	String
b	This field is used to store the book name.	String
p	This field is used to store the price of the book.	Double
q	This field is used to store the quantity (sold) of the book.	Integer
bt	This field is used to store the book type (Regular book or Text book)	String
opp	This field is used to store the original publisher price of the book.	Double

GR	This field is used to store the growth rate.	Double
publishYear	This field is used to store the year of book published.	String

3.2. Hadoop MapReduce

Following table documents various MapReduce tasks mentioned in the workflow:

Map Reduce Task 1	This task will be the entry point of the price calculation process. It will calculate the "η(eta)" value i.e. the price elasticity of demand based on the raw data.
Mapper 1_Elasticity.java	**Input:** DM::BookID::b::p::q::bt::opp::GR::publishYear
Reducer 1_Elasticity.java	**Output**: Key = BookID Value = DM_b_p_q_bt_opp_GR_e_publishYear
Map Reduce Task 2	This task will predict an estimated price for future Year Month
Mapper2_Forecast.java	**Input:** BookID list(DM_b_p_q_bt_opp_GR_e_publishYear)
Reducer2_Forecast.java	**Output:** Key = BookID Value = DM_b_p_q_bt_opp_GR _e_p'_boolC_ publishYear
Map Reduce Task 3	This task will calculate the forecasted price factor and the depreciated price factor.
Mapper3_IndvPrices.java	**Input:** BID list(DM_b_p_q_bt_opp_GR _e_p'_boolC_ publishYear)
Reducer3_IndvPrices.java	**Output:** Key – BID Value - DM_b_p_q_bt_opp_GR _e_p'_boolC_ publishYear _P12_P3
Map Reduce Task 4	This task will calculate the weight for forecasted price factor and weight for depreciated price factor based on which the ideal price for the book is calculated for that year month.
Mapper4.java	**Input:** BID list(YM_BN_P_Q_BT_OPP_GR_e_p'_boolC_PY_P12_P3)
Reducer4.java	**Output:** Key – BID Value – YM_BN_P_Q_BT_OPP_GR_e_p'_boolC_PY_P12_P3_w12_w3_pIdeal

DM – Data (Year) Month, BookID – ISBN, b – Book Name, p – Price, q – Quantity, bt – Book Type, opp – Original Publisher Price, GR – Growth Rate, publishYear – Publishing Year, e – η (eta), p' – Price predicted by estimation for future Year month, boolC – Flag for current year, P12 – Forecasted price factor, P3 – Depreciated price factor , w12 – weight for forecasted price factor, w3 - weight for depreciated price factor, pIdeal – Ideal Price

Other Classes

Program	Contains the main method and the job chaining for all for Map Reduce tasks.
BookEntry	This class acts as a data parser for the input book data.
ForecastPrediction	This class contains methods to forecast prices for trending months based on baseline data.

3.3. Test Data Creation

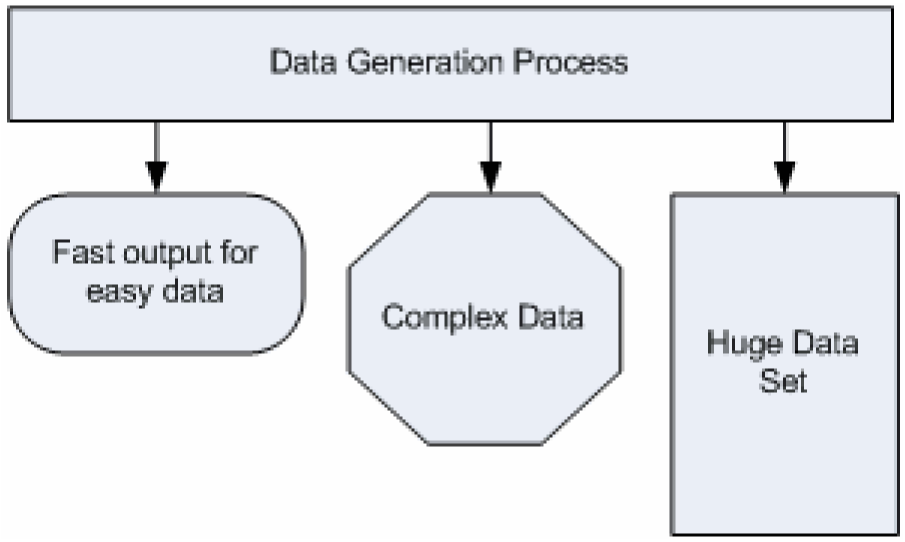


Figure 5 Data generation process

We have created an *excel file* template that takes input required in the form of aggregates and creates a syntactically and semantically correct query required to load data in MongoDB. Care is taken to make sure it automatically validates queries generated before executing on local MongoDB client (called MongoChef). Successful execution of this query will create necessary aggregates inside collections defined in MongoDB.

In accords of iterative methodology being followed to complete this project, even test data was generated in rounds. Unit testing data was made available during first iteration for ease of development and quick result verifications. This helped reduce dependencies during initial stages.

During the first round of system testing, a complex data (with wild characters and null value records) was created to test robustness of system . Final implementation testing in Iteration 2 was done using a data-load for stress testing.

4. Testing

4.1. Test bed creation

Given the minimal user interaction with system, most test cases written were to check for fidelity of numbers calculated. Once the system was ready with integration between MongoDB and Hadoop map-reduce framework, two types of tests were carried in various rounds

4.1.1. Smoke Test

This was to ensure overall correctness of integration of different system components. Subsets of test cases that cover most important functionality were executed. Smaller dataset used led to quicker execution revealing simple failures. These bugs have been fixed and system was passed for entire System test.

4.1.2. System Test

Complex and bigger datasets were used to perform all test cases. Major performance concerns were logged and tackled in reducer programs. Primarily forecast calculation seemed to be an efficiency bottleneck. Modularizing it's codebase and implementing an entirely different Forecast_Prediction.java led to quicker estimates and swift results were hence observed.

5. Conclusion

Estimating price elasticity and ideal prices for books can be leveraged by top executives heavily to get quicker and first-rate insights. With this knowledge readily available, a lot of managerial decisions can be taken that lead to increase in sales revenue. Knowing revenue trend can definitely help firms/publishers manage book inventories beforehand based on seasonality depicted in output graphs. They can initiate efforts in targeted sales and advertising campaigns narrowed down for certain low-performing books to bump its sales. Since we will be working on this application in future to develop a large-scale product and deploy it on online cloud (Azure), there are several more components that will be added. We believe introducing a social aspect viz making book available for free/minimal prices beyond its predefined age will help increase traction of this application. Moreover, being able to factor results based on different location will be a customary addition so that regional managers can view relevant dashboards and heat maps based on how selected book is foreseen to perform in different geographical locations. There is no doubt this project has definitely laid the required foundations and much more.

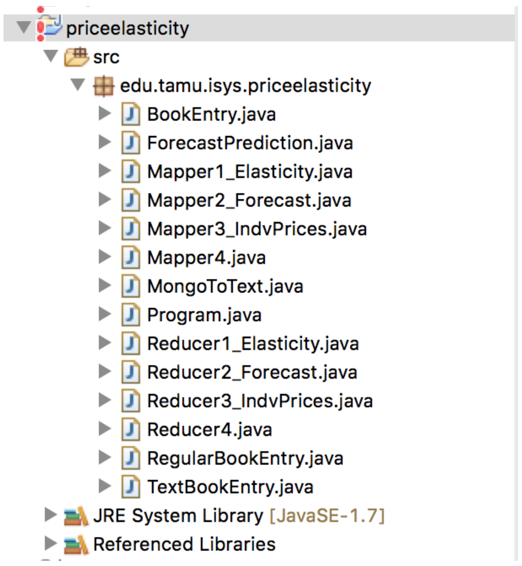
Being our first NoSQL and MapReduce implementation, it came with a plethora of learning possibilities. Every team-member has made the most of it and this experience definitely augments our career ambitions.

6. References

1. Jangjaimon, I., & Tzeng, N. Effective Cost Reduction for Elastic Clouds under Spot Instance Pricing Through Adaptive Checkpointing. *IEEE Transactions on Computers IEEE Trans. Comput.*, 396-409.
2. Reimers, I., & Waldfogel, J. Throwing the Books at Them: Amazon's Puzzling Long Run Pricing Strategy. *SSRN Electronic Journal SSRN Journal*.
3. The elasticity of demand for books, resale price maintenance and the lerner index. Retrieved December 7, 2015, from http://www.opengrey.eu/item/display/10068/97075
4. Price- and Cross-Price Elasticity Estimation using SAS. Retrieved December 7, 2015, from http://support.sas.com/resources/papers/proceedings13/425-2013.pdf
5. Hwang, S., & Kim, S. Dynamic Pricing Algorithm for E-Commerce. *Advances in Systems, Computing Sciences and Software Engineering*, 149-155.
6. “Current Data Security Issues of NoSQL Databases”, https://www.fidelissecurity.com/files/NDFInsightsWhitePaper.pdf

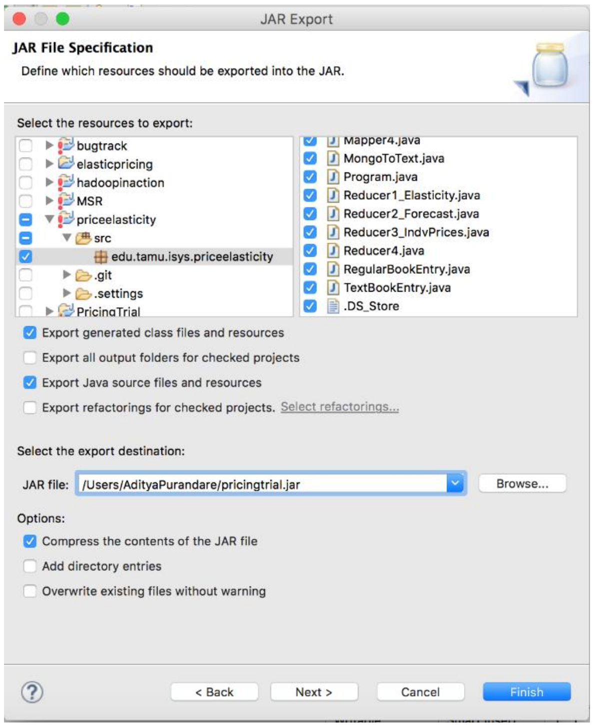
Appendix A: Standard operating procedure

1) Make sure your codebase is present in folder under package edu.tamu.isys.priceelasticity



```
▼ priceelasticity
  ▼ src
    ▼ edu.tamu.isys.priceelasticity
      ▶ BookEntry.java
      ▶ ForecastPrediction.java
      ▶ Mapper1_Elasticity.java
      ▶ Mapper2_Forecast.java
      ▶ Mapper3_IndvPrices.java
      ▶ Mapper4.java
      ▶ MongoToText.java
      ▶ Program.java
      ▶ Reducer1_Elasticity.java
      ▶ Reducer2_Forecast.java
      ▶ Reducer3_IndvPrices.java
      ▶ Reducer4.java
      ▶ RegularBookEntry.java
      ▶ TextBookEntry.java
  ▶ JRE System Library [JavaSE-1.7]
  ▶ Referenced Libraries
```

2) Export jar along with dependencies (just the ones needed to execute any Hadoop program) into a file called 'priceelasticity.jar'



JAR Export

JAR File Specification

Define which resources should be exported into the JAR.

Select the resources to export:

```
☐ ▶ bugtrack                         ✓ Mapper4.java
☐ ▶ elasticpricing                   ✓ MongoToText.java
☐ ▶ hadoopinaction                   ✓ Program.java
☐ ▶ MSR                              ✓ Reducer1_Elasticity.java
▬ ▼ priceelasticity                  ✓ Reducer2_Forecast.java
▬   ▼ src                            ✓ Reducer3_IndvPrices.java
✓       edu.tamu.isys.priceelasticity ✓ Reducer4.java
☐   ▶ .git                           ✓ RegularBookEntry.java
☐   ▶ .settings                      ✓ TextBookEntry.java
☐ ▶ PricingTrial                     ✓ .DS_Store
```

- [✓] Export generated class files and resources
- [ ] Export all output folders for checked projects
- [✓] Export Java source files and resources
- [ ] Export refactorings for checked projects. Select refactorings...

Select the export destination:

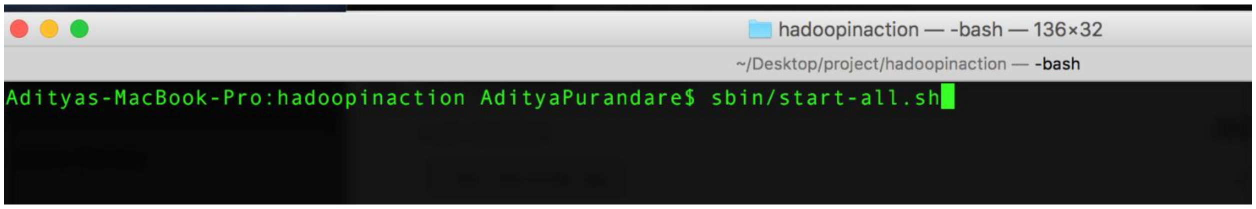
JAR file: /Users/AdityaPurandare/pricingtrial.jar   Browse...

Options:

- [✓] Compress the contents of the JAR file
- [ ] Add directory entries
- [ ] Overwrite existing files without warning

3) Start hadoop cluster using the following command on terminal

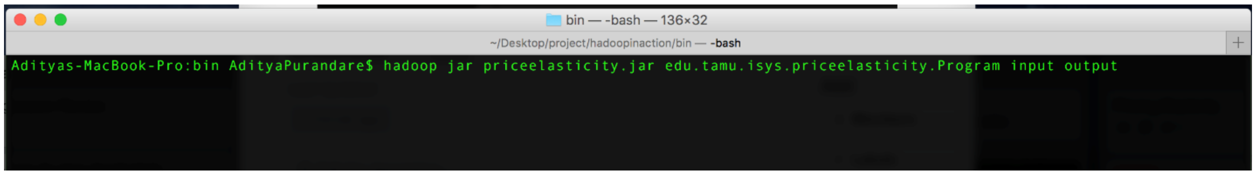
sbin/start-all.sh



```
hadoopinaction — -bash — 136×32
~/Desktop/project/hadoopinaction — -bash
Adityas-MacBook-Pro:hadoopinaction AdityaPurandare$ sbin/start-all.sh
```

4) From the terminal execute following command to run the program

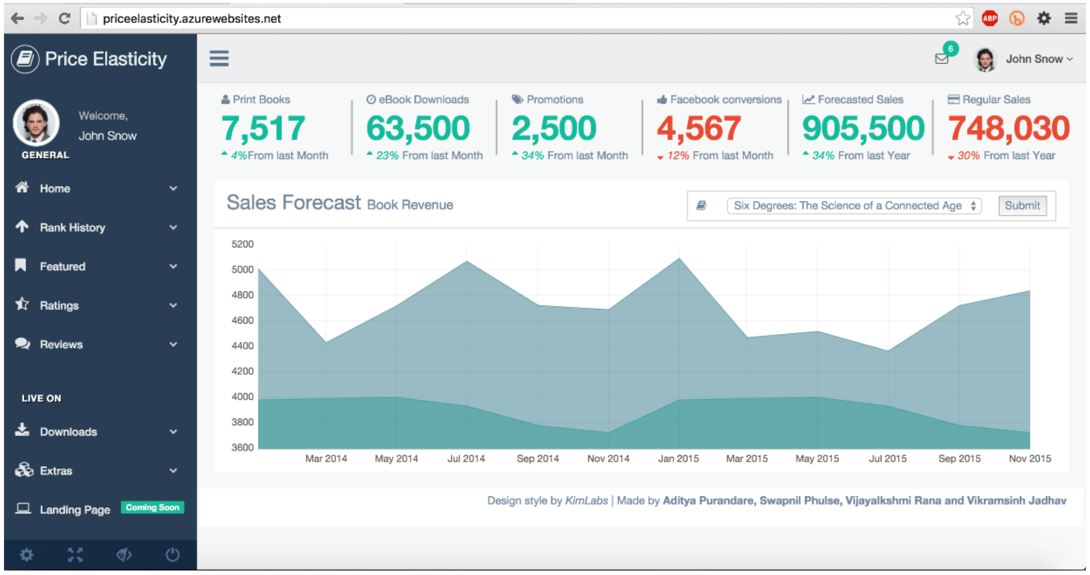
hadoop jar priceelasticity.jar Program input output



```
bin — -bash — 136×32
~/Desktop/project/hadoopinaction/bin — -bash
Adityas-MacBook-Pro:bin AdityaPurandare$ hadoop jar priceelasticity.jar edu.tamu.isys.priceelasticity.Program input output
```

5) After successful execution of program a live webpage (hosted at http://priceelasticity.azurewebsites.net/) should open that shows the sales revenue based on original prices and ideal estimated prices.

Note – Please make sure you use GOOGLE CHROME as web browser (currently supported)


priceelasticity.azurewebsites.net
Price Elasticity
John Snow
Welcome, John Snow
GENERAL
Home
Rank History
Featured
Ratings
Reviews
LIVE ON
Downloads
Extras
Landing Page
Coming Soon
Print Books
7,517
4% From last Month
eBook Downloads
63,500
23% From last Month
Promotions
2,500
34% From last Month
Facebook conversions
4,567
12% From last Month
Forecasted Sales
905,500
34% From last Year
Regular Sales
748,030
30% From last Year
Sales Forecast Book Revenue
Six Degrees: The Science of a Connected Age
Submit
5200
5000
4800
4600
4400
4200
4000
3800
3600
Mar 2014
May 2014
Jul 2014
Sep 2014
Nov 2014
Jan 2015
Mar 2015
May 2015
Jul 2015
Sep 2015
Nov 2015
Design style by KimLabs | Made by Aditya Purandare, Swapnil Phulse, Vijayalkshmi Rana and Vikramsinh Jadhav