**San Diego House Price Predicting**

**Wen Yan, Mengting Wang, Salah Ahmad, Xia Song**

# **Summary of Scalability and Robustness Requirements**

## **Scalability Requirement**

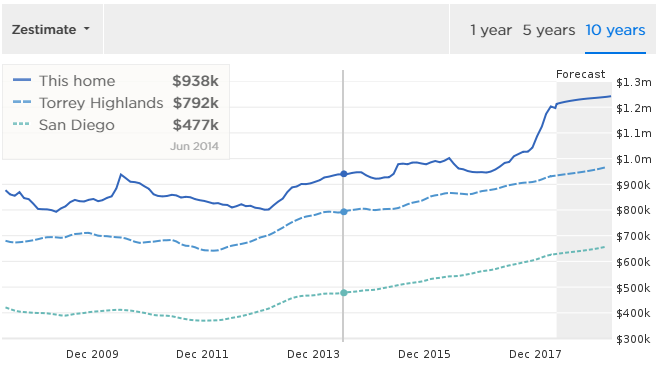
* **More data - Spatial and Temporal Scalability**

**Scale vertically (Spatial):** SD county => state => whole country

****

**Scale horizontally (Temporal):** 2015~2017 => all historical housing market

We’re not just modeling and evaluate each individual house price under current market, but also want to evaluate the price for all time long. Historic estimates give insight how a home (or an area) has changed in value over the years. Below figure shows Zillow Zestimate all time long evaluation, we want to do similar.



* **More dataset(from other source)**

We can imagine that there are many other valuable data sources that can help improve the modeling accuracy, eg. demographic, various census and realtor and buyers’ comment, pictures of the house etc. If new data source still fit relational data model, it’s naturally scaling of our current relational data solution. Some of them especially multimedia may not be suitable with relational database anymore, in this case a hybrid database solution may be necessary.

* **Modeling scalability**

**Modeling computation**

To achieve the estimation for more houses(of whole country) and cover longer time period, modeling computation need be scaled accordingly. But our modeling strategy or algorithm doesn’t change, which means one model will be built for each county to predict the evaluation price of the houses in just that county, it’s not using whole country’s data to build one single model to predict all houses. Same thing for time period, we just use data of current market eg. 4 months for training to predict the price in next 4 months.

In another word, although dataset scales, each model computation requirement doesn’t change, and we just build more models for more areas and periods. Based on this scalability requirement, we’re flexible on choosing the computation scaling strategy. We’re not limited to distributed computing technics eg HDFS+Spark/MapReduce etc, task distribution infrastructure may better serve our purpose. This is illustrated in approach section.

* **Modeling frequency**

There’s high chance that modeling algorithm will be improved from time to time, and more features from more data sources can be added. Whenever this happens, the model need be re-trained and predicted on the whole dataset. From Zillow’s experience, they undergo 3 major updates on modeling algorithm (2006, 2008 and 2011) since 2006, and many minor updates in between. So roughly, a full regression is necessary once per 1 or 2 years.

## **Robustness Requirement**

**Modeling evaluation**

In order to determine the performance of our model in predicting the target on new and future data, we need some matrix to evaluate our model. The evaluation metrics depends on the model category, for example, regression model, binary classification model and multiclass classification model. For our project, we use Random Forest regression model to predict house price. There are different evaluation metrics available, such as Mean Absolute Error, Root Mean Absolute Error, Relative Absolute Error, Relative Squared Error, and the Coefficient of Determination. The “error” means the difference between the difference between the predicted value and the true value. We adopted Root Mean Absolute Error to measure our model’s performance.

**Fallback plan for outliers and NAN data**

Missing values and outliers are frequently encountered while collecting and processing data. The presence of missing values lead to a smaller sample size, and it can also produce biased results. As a part of the pretreatment process, missing data are either ignored in favor of simplicity or replaced with the substituted values estimated with statistical method.

We view noises in data as two types:

Type 1: True noises such as transactions that don’t have enough values or have missing values on important features such as school, etc.

Type 2: Reasonable outliers such as foreclosure transactions that appear to be much lower price than zip average

In general, both types of noises are treated separately from normal transactions, with different modeling strategies. For type 1 noise, we try to fill in missing features with aggregated values such as zip average. In the case where certain zip codes don’t have enough transactions, filling missing values in with zip average wouldn’t make sense anymore as this approach will make the model extra sensitive and skewed to outliers; in cases like this, we use engineered features such as zip clusters to perform modeling on zip cluster level. This is not perfect as by doing this, zip specific attributes get lost in the middle of aggregation. However it can be a certain level of remedy that alleviate missing value problem. For type 2 noise, we simply fit those transactions into a separate model as foreclose transactions behave differently than normal transactions.

## Data Pipeline and Process

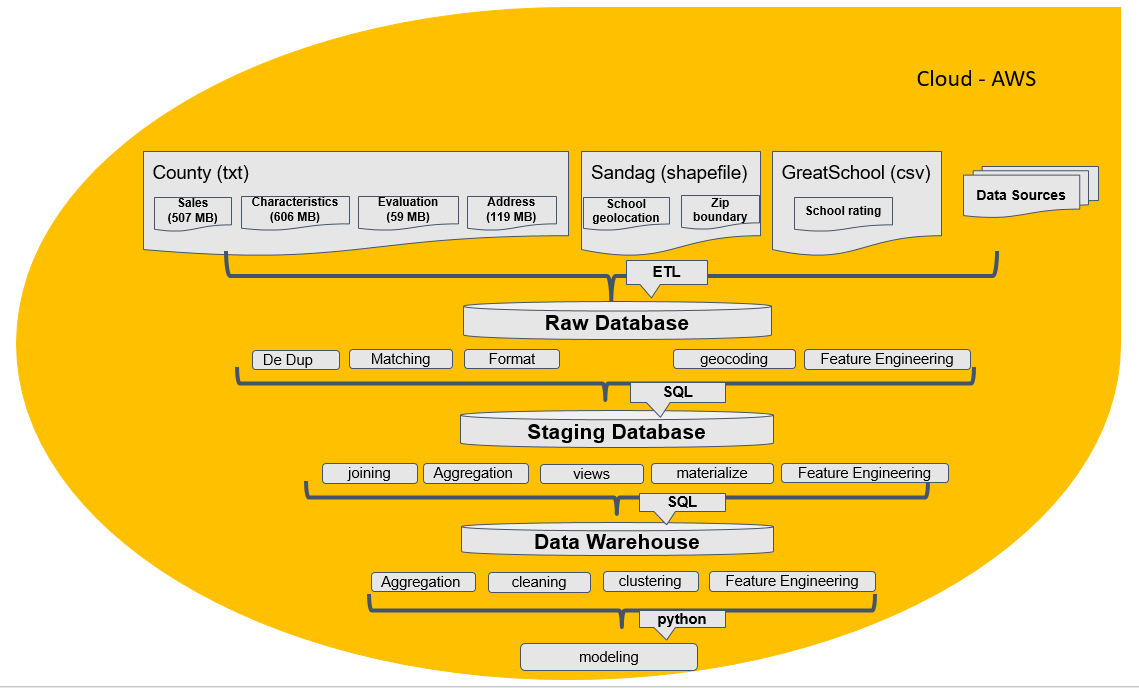


Fig 2. Data processing flow chart and data architecture

# Approach

## Scalable Data Management to Model Simulation

* Database Storage/Engine:
  + We choose an RDMS to store our data. At this point it is in PostgreSQL.
  + We use standard SQL DDL and DML. This will enable us to move to a high-end enterprise RDMS such as SQL Server or Oracle when needed.
* Optimization for scalability:
  + Denormalized entities were designed to enable faster querying.
  + Performance tuning best practises such as indexes, partitioning, etc… will be implemented.
* Deployment:
  + Current and short term needs (i.e. size, querying, etc…) is working well for on-premise deployment.
  + For the sake of scalability readiness for massive growth (i.e. going national or international), we will deploy the solution on the cloud (AWS).

### Data Collection

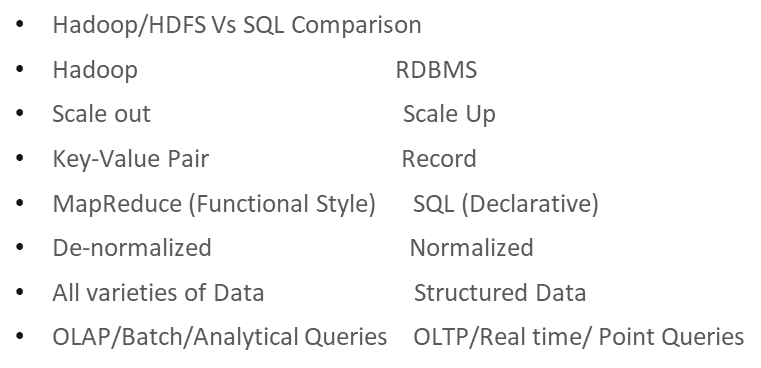
Given the data source of our project totally depends on government agency, we can not get all the data at one time. Therefore data organizations and management is one big challenge for our project. In order to improve management process and enhance data using experience, the following steps are adopted to process and store data (Fig 2).

1. Collecting source data into raw data base from government agency and other agencies, source data include house sale historical data, house characteristics data, house location data, house school district geolocation data and house school rate data;
2. Process raw data into our staging database through deduping, matching, uniform data format, geocode house location, meanwhile based on raw data, we also did feature engineering.
3. Base on data management and data organization rules, like data join, data aggregation, create data views, data materialize and feature engineering, finally store data into data warehouse.
4. In order to serve model simulation, when we extract data from our data warehouse, we did data aggregation, data cleaning, data clustering and data feature engineering.

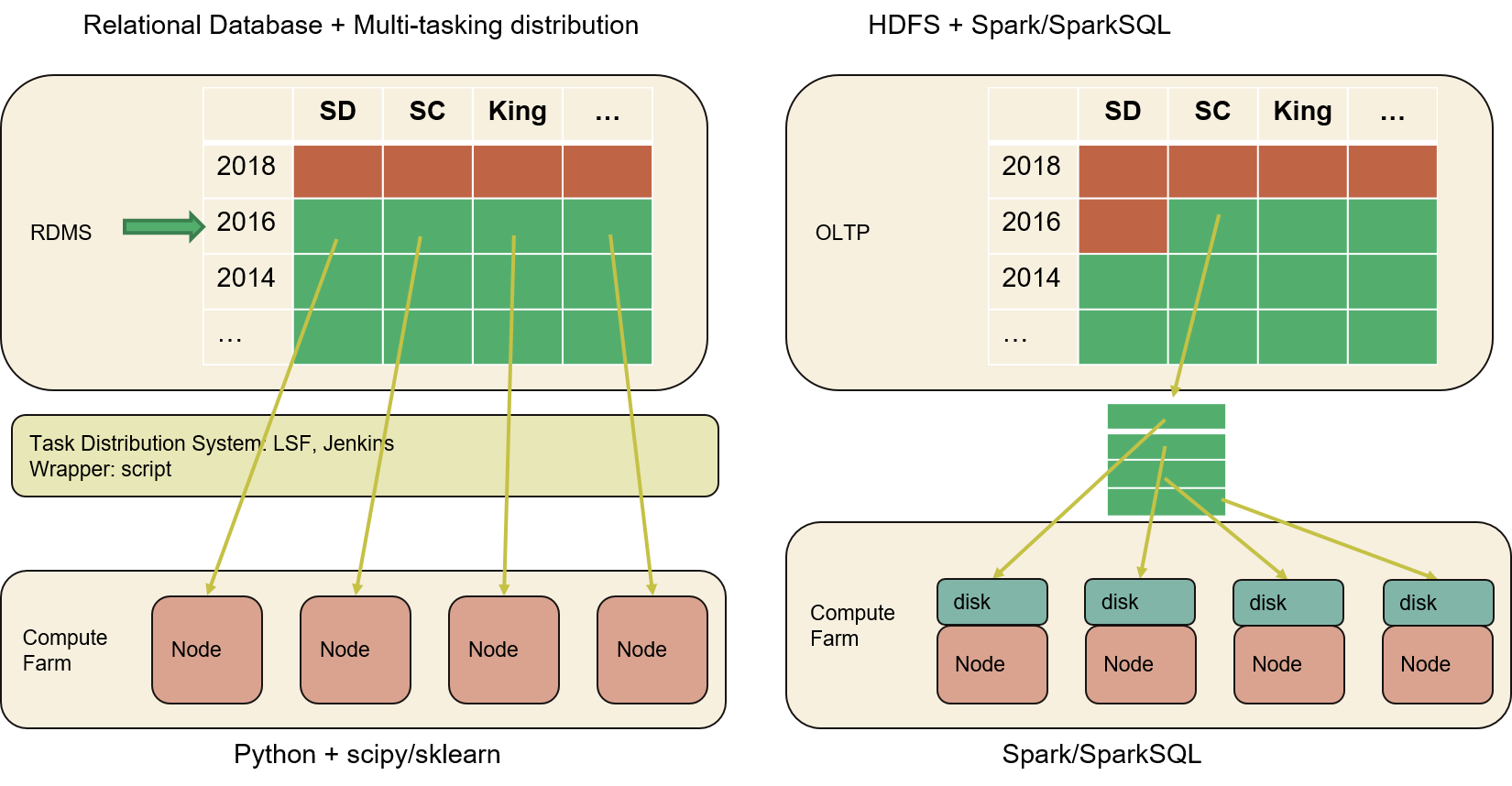
## **Programming(Computation) Model**

As described in scalability requirement section, our modeling scalability is multi tasking rather than distributed computing, we have the flexibility to leverage between distributed-computing techniques and multitasking techniques

Hadoop stack is the ad-hoc distributed computing technic. A typical setup for distributed-computing model is HDFS + Spark/MapReduce. And there are also a lot of mature solutions and infrastructure for multi-tasking computing as well, in both commercial and open-source world. As the computation model is highly bound with data model, distributed computing usually are coupled with distributed files system which is preferred with high throughput and low latency tasks, while central database like RDMS is more suitable for both application and analytics as ACID is more strictly followed as well as high throughput. Below is a brief comparison between RDMS and HDFS.



Below is a brief side-by-side comparison of the two architecture that deployed on our problem.



Brief review of our requirement for data and programming model

* Our dataset are tabular relational data and some geo data, and they can scale up to TiB.
* Data query are needed for both application and analytics. Data query need be application friendly and have enough throughput for analytics.
* A lot of virtual/materialized views and joining/aggregations for feature engineering
* Geocoding functions are better to be provided by database server.
* Modeling is built with data segment, model scaling means more building models, so the scalability requirement is distributed tasking rather than distributed computing.

From above list, RDMS (AWS RDS) is good fit to our requirement, and HDFS/Spark model has various hurdles to overcome, below are some top items.

* OLTP database must be SQL which is determined by application and the nature of our raw data. So if we choose HDFS as OLAP data warehouse model, extra effort need be taken for the conversion.
* To really take advantage of distributed computing, the data need be partitioned very carefully to nodes to avoid data shuffling during various MapReduce-style operations.
* A lot of feature engineering with joining/aggregation are done by SQL database server, migrating these functions to Spark takes quite an effort of development and maintenance. Although SparkSQL can alleviate the pain, but still its performance is not competent to SQL DB engines, and it’s lack of indexing and various joining schemes.
* Advanced modeling strategy like segmentation and dynamic sliding window for training make things more complicated to make data partitioned onto each node’s local disk.

So generally, RDMS like AWS RDS service provides all the scalability we required for data modeling, and computation can be more easily achieved with task distribution system since our modeling strategy doesn’t require all data for each training.

# **Evaluation Strategy and Plan**

**Database Server**

We want make sure the database server not always overloaded or under utilized. Overloading means database turns to be bottleneck. Could be either bound on throughput or computation. For either case, AWS RDS can be easily scaled with more resource until hitting upper limit.

AWS RDS performance monitor is a good tool to keep tracking performance of the database server regarding to various metrics, I/O throughput, latency, CPU and memory. We don’t need extra effort to develop anything else for the evaluation.

**Modeling performance**

Modeling speed is not so critical to analytics part of our project, as long as it’s in minutes for single model training and prediction (eg 12 months data training and 4 months data testing). And for full regression on whole historic dataset, it’s acceptable to take hours. Right now one full regression takes ~ 1 hour to finish San Diego 35 years worth data. We need make sure the node number we added are linear to the data amount that is scaled, and the total hours doesn’t change much.

**Team Contribution:**

* **Salah Ahmad:** Data Management, Pipeline and Scalability approach and revision.
* **Wen Yan:** Programming Model architecture selection, solution validation, and modeling regression across whole history.
* **Mengting Wang:** Scalability requirements, robustness evaluation and visualization plan; continue to make progress on visualization and demo tool build-up.
* **Xia Song:** Worked on initial paper draft.