

## Using machine learning for intelligent shard sizing on the cloud

Narayanan Venkateswaran<sup>1</sup>, Anurag Shekhar<sup>2</sup>, Suvamoy Changder<sup>3</sup> and  
Narayan C. Debnath<sup>4</sup>

<sup>1</sup>Department of Computer Science and Engineering, National Institute of Technology Durgapur, India

<sup>2</sup>MySQL Oracle India Pvt Ltd., India

<sup>3</sup>Department of Computer Science and Engineering, National Institute of Technology Durgapur, India

<sup>4</sup>Department of Software Engineering, Eastern International University, Vietnam

---

### Article Info

Received Dec 31<sup>st</sup>, 2018

---

#### Keyword:

Machine Learning  
Sharding  
Horizontal Partitioning  
Cloud  
Server Sizing  
Deployment Planning  
Resource Allocation  
Data Sizing

---

### ABSTRACT

Sharding implementations use conservative approximations for determining the number of cloud instances required and the size of the shards to be stored on each of them. Conservative approximations are often inaccurate and result in overloaded deployments, which need reactive refinement. Reactive refinement results in demand for additional resources from an already overloaded system and is counterproductive.

This paper proposes an algorithm that eliminates the need for conservative approximations and reduces the need for reactive refinement. A multiple linear regression based machine learning algorithm is used to predict the latency of requests for a given application deployed on a cloud machine. The predicted latency helps to decide accurately and with certainty if the capacity of the cloud machine will satisfy the service level agreement for effective operation of the application. Application of the proposed methods on a popular database schema on the cloud resulted in highly accurate predictions. The results of the deployment and the tests performed to establish the accuracy have been presented in detail and are shown to establish the authenticity of the claims.

---

#### Corresponding Author:

Suvamoy Changder,  
Department of Computer Science and Engineering,  
National Institute of Technology Durgapur,  
Mahatma Gandhi Avenue, Durgapur 713209,  
West Bengal, INDIA  
Email: [suvamoy.nitdgp@gmail.com](mailto:suvamoy.nitdgp@gmail.com)

---

### 1. Introduction

Any increase in the number of users of an application causes the data stored and used by the application to increase. This raises the demand for storage and processing resources. Since the cloud allows for such on-demand scaling of resources, it is a popular choice for similar applications requiring elastic backends [16]. Sharding topologies split the database into multiple shards and store one or more shards in each cloud instance [11] [18]. Resharding operations, shard splits and merges, are done on demand by leveraging the elasticity provided by the cloud instances. Several datastores with successful and popular sharding implementations have added elastic extensibility on the cloud [6] [26].

Sharding implementations distribute the data uniformly across the set of available servers [23] [27]. If the uniform distribution causes workload in excess of server capacity, the amount of data stored is refined reactively.

Reactive refinement is often performed after the load spike is detected on a shard. Refining shards imposes an additional workload to the system, and can prove to be inefficient.

This paper proposes an application specific, machine learning based partitioning scheme as an alternative for the uniform distribution of data across servers. The methods proposed in this paper allow for accurate server sizing and reduce the need for reactive refinement in an application specific sharding solution. Unlike the existing solutions, the proposed solution attempts to create accurate partitions from the beginning by learning from empirical and existing data.

section 2 talks about existing sharding solutions, their use of reactive refinement and the associated problems. section 3 on page 4 proposes the predictive sharding scheme. The proposed method is tested quantitatively in section 4 on page 7 and it is shown that the proposed methods help create efficient shards, given a Service Level Agreement (SLA) for application latency.

## 2. Background and Applicability

This section begins with a brief introduction to the distributions created by popular sharding setups. The need for reactive refinement, its downsides and its alternatives are also discussed in detail. This is followed by an explanation for SLA, its implication for applications and the motivation behind relating it to shard sizes and other database and system parameters.

### 2.1. Background

Existing sharding implementations begin with a conservative estimate of the number of cloud server instances. The data items are distributed across the servers by using consistent hashing [5] [19] [2]. Consistent hashing maps the sharding key to a server in the topology. This is accomplished by using a hashing function that operates on the sharding key. The range of values of the hash function is referred to as a hash ring. For example, MD5 is a popular hashing function that forms a ring of values from 0 to  $2^{128} - 1$ .

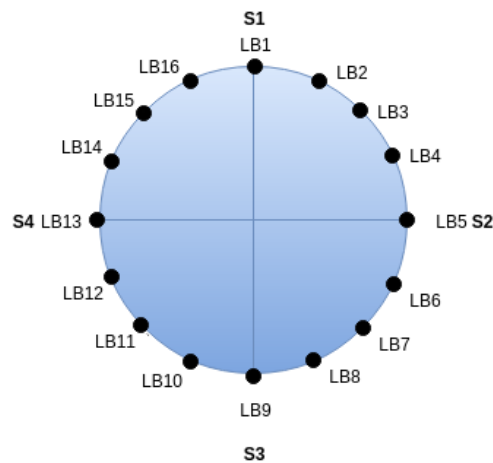


Figure 1: Creating Partitioning Buckets

A data item is mapped to a position on the MD5 hashing ring by applying the MD5 hashing function on the sharding key of the data item. Consider, for example, an employee schema that is sharded on employee ID. A data item with an employee ID as EMP\_ID is mapped to a position in the hash ring by using the value of  $md5(EMP\_ID)$ . Thus each data item is being mapped to a unique position on the hash ring.

The consistent hashing implementation divides the MD5 hash ring [20] into multiple equal size ranges as shown in fig. 1. Each of these ranges is referred to as a virtual partition. Equal number of virtual partitions are distributed to each server in the topology.

In fig. 1 the hash ring is divided into 16 virtual partitions. The hashing starts with a conservative estimate of 4 servers. Each server is associated with four virtual partitions.

Consistent hashing distributes the data uniformly across the servers. This is achieved by virtue of the randomness associated with the hashing function [9] [20]. The randomness property ensures an equal probability of selection for each shard, thus ensuring a uniform distribution over them.

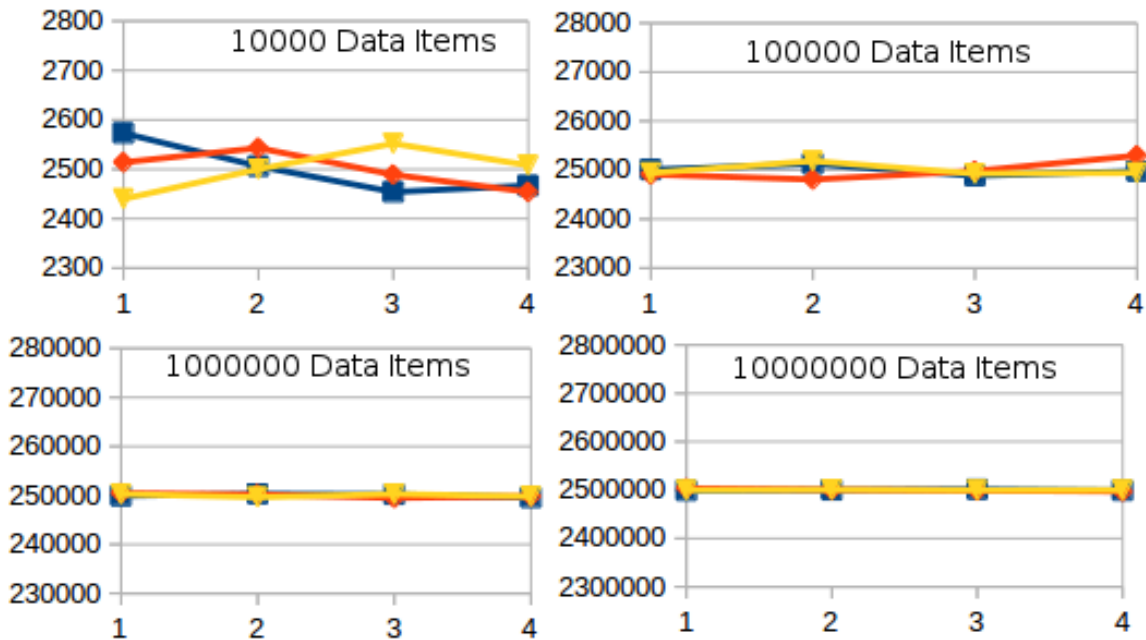


Figure 2: Distribution Using Consistent Hashing

The uniform distribution created is independent of the type or value of the sharding key. Fig. 2 shows four graphs. The graphs show the distribution of 10000, 100000, 1000000, 10000000 items in the topology represented in fig. 1. Each graph contains three lines, for three different types of keys. The distribution was attempted on integer, string and random keys and the data was plotted as a line graph for each key count. It can be seen from this graph that consistent hashing creates a uniform distribution in each case. It can also be seen that distribution becomes more uniform as the number of data items increases.

However, uniform distribution is often not ideal and can cause load in excess of what can be handled in a shard. This can result in overloading the shard server. In such an event the amount of data stored is refined reactively.

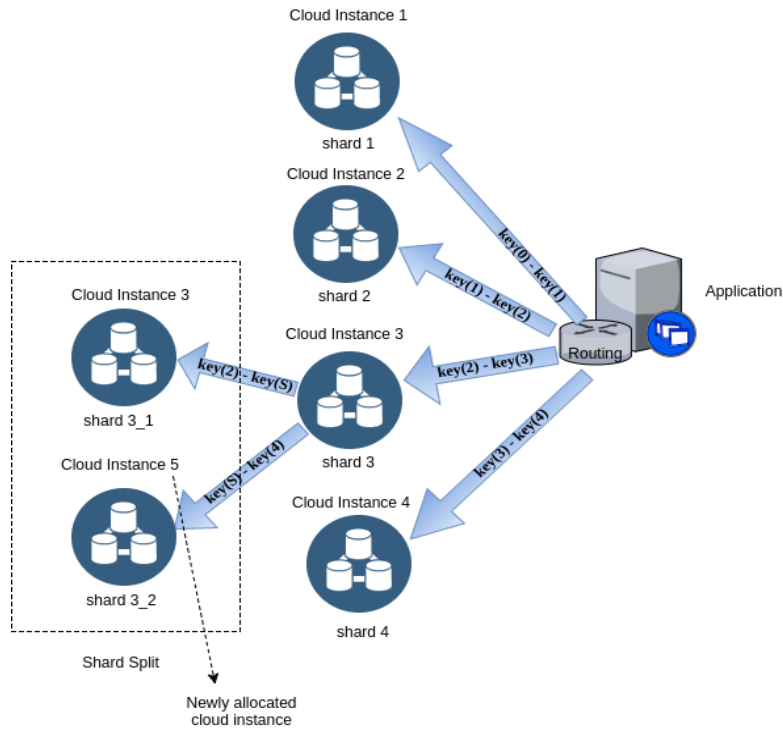


Figure 3: Sharding on the cloud.

fig. 3 shows an example of reactive refinement in the form of a shard split operation. The initial shard setup contains four shard key ranges, where  $K$  represents the key of a data item that is resolved to one of the shard .

1. Shard 1 –  $[\text{key}(0), \text{key}(1))$
2. Shard 2 –  $[\text{key}(1), \text{key}(2))$
3. Shard 3 –  $[\text{key}(2), \text{key}(3))$
4. Shard 4 –  $[\text{key}(3), \text{key}(4))$

The shard 3 is split into two new shards,

- Shard 3\_1 –  $[\text{key}(2), \text{key}(S))$
- Shard 3\_2 –  $[\text{key}(S), \text{key}(3))$

The range  $[\text{key}(S), \text{key}(3))$  is moved into a new cloud instance. Application logic is used to route queries to the shard containing the key range relevant to the query. For example, if a query is fired for a data item whose key value  $K_1$  is in the range  $\text{key}(1) < K_1 < \text{key}(2)$ , the query is routed to shard 2.

Fine granularity reactive refinement was discussed in detail in the E-Store framework [22] [4]. There has been active research on predictive provisioning. P-Store [23] proposes a dynamic algorithm for predicting load and automatically scaling the servers. Although P-Store addresses the problem of reactive partitioning during overloads, it does not address the problem of disproportionate allocation of data caused by uniform distribution. Thus, the actual cause of reactive partitioning is not addressed by P-Store.

## 2.2. Applicability

Applications serve responses within a predetermined SLA. Addressing requests within the SLA is of prime importance, especially for online services. An online shopping website could lose customers if it takes too much time to respond to customer requests.

Uniform distribution of load does not always result in accurate partition sizes and desired SLAs. Existing sharding implementations reactively refine the uniform distributions to produce accurate shard sizes.

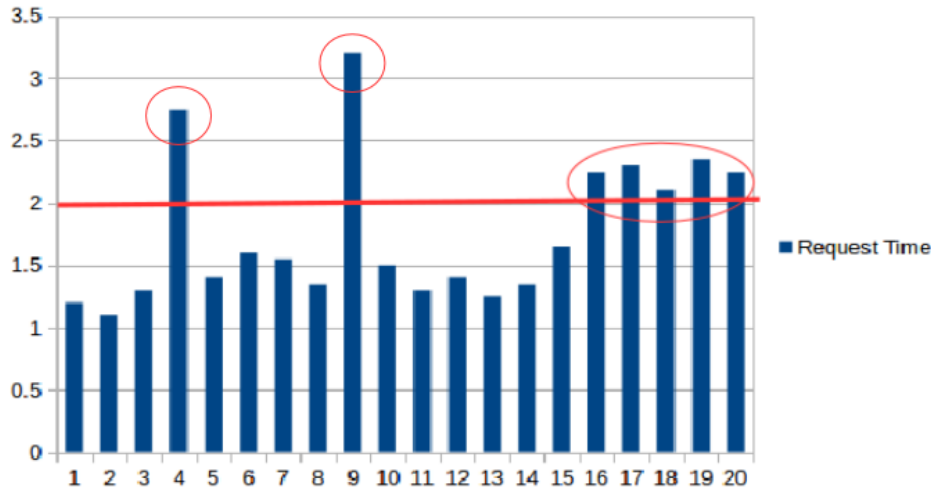


Figure 4: Response Times

Fig. 4 shows the response times of such an application with SLA violations circled in red [15]. Such SLA violations trigger repartitioning. Reactive partitioning of this nature places additional demand for resources on an already overloaded system. However, if it were possible to predict appropriate sizes for a given cloud instance, the number of reactive refinements could be drastically reduced.

Consistent hashing in itself does not take into account the performance of an application for a given shard size on the target server. Assuming uniform load, the maximum size of the data that can be deployed on a given server instance while still managing to meet SLA expectations depends on several parameters. These include application specific metrics, server capacity, the promised SLA, etc. By correlating these parameters with the latency metrics from the application, the appropriate data size can be determined. Since the parameters are application specific and vary with each deployment, the method proposed must be generic enough to be applied to any setup. Also, there might be several parameters that have different degrees of influence. It can be very difficult to determine, manually, the exact relationship between these parameters and their influence on shard size.

In the next section we consider how all the parameters influencing application performance can be taken into account and their degree of influence be accurately determined for correct shard size prediction.

### 3. Proposed Implementation

#### 3.1. Intuition

When an application runs on a given server configuration, there are several parameters that affect the response time of the application. The parameters can be broadly classified into,

- System Level - e.g. RAM, CPU cores, etc.
- Application Level - e.g. Data size, Buffer Size, Etc.

Let the latency be represented by  $Y$ , while the system level and application level parameters are given by RAM ( $X_1$ ), CPU ( $X_2$ ), Data size ( $X_3$ ) and Buffer size ( $X_4$ ). The relationship between latency and a system / application level parameter is represented by eq. (1).

$$Y \propto X_1 \quad (1)$$

This can in turn be expressed as,

$$Y = c_1X_1 + b_1 \quad (2)$$

In eq. (2)  $c_1$  represents the weight of the parameter and can be interpreted as a quantitative representation of the influence of the parameter  $X_1$  on the latency. The larger the weight, the larger the influence of the parameter on the latency.

A summation of eq. (2) over all the system and application level parameters can be represented by eq. (3),

$$Y = c_0 + c_1X_1 + c_2X_2 + c_3X_3 + c_4X_4 \quad (3)$$

$Y$  is a dependent variable, while  $X_1, X_2, X_3, X_4$  are referred to as independent variables. Thus, eq. (3) predicts the latency, assuming RAM, CPU, Data size and Buffer size are known. This equation forms the underlying principle of a multiple linear regression model [24] [13].

### 3.2. Building the prediction model

The prediction model is built from metrics collected by running the applications on heterogenous cloud configurations. The metrics collected can be classified into:

1. System Metrics - Collected from the cloud machine on which the application is deployed
2. Application Metrics - Collected through observing application performance

The system metrics involve parameters like RAM and CPU usage, collected during application operation. The application metrics involve metrics like latency, data size, writes, reads, etc. The collection is done over multiple data points ( $> 100$ ) and averaged to even out spikes. The application and the collection process are explained in greater detail in section 4 on page 7.

The collected metrics are used to form training and test sets using the K-fold cross-validation method. K-fold cross-validation divides the dataset into K partitions and uses each in turn for testing. When one of the partitions is used for testing the rest are used for training. Repeating this K times ensures that all the partitions are used for testing. From this, the best-fitted parameters are used to build and validate the model. [1] [12].

### 3.3. Validating the prediction model

After collecting data, the parameters collected need to be analyzed to identify the most relevant ones for regression analysis. The following statistical metrics are used to perform backward elimination on the parameters to eliminate the irrelevant ones,

- P-Value: The p-value is used to determine the significance of the results. It also helps determine if there is a linear relationship between the dependent and the independent variables at a given level of significance. [10] [21] [17].

The p-value is used to perform a t-test on each parameter and to determine if there is sufficient evidence of a linear relationship between the dependent and the independent variables at the 5% level of significance [10] [8].

- Adjusted R-squared: The adjusted R-squared helps determine if a given parameter improves the model more than expected by chance. The value of the adjusted R-square for the model increases with the addition of a parameter that improves the model. The value decreases otherwise. [10] [21] [17].
- Variance Inflation Factors (VIFs): Multi-collinearity between parameters in the regression model can be determined using the VIFs. A large value for the VIF (greater than 10) implies that two parameters

in the regression model are multi-collinear and can be linearly predicted from each other. The presence of both of the parameters in this case leads to biasing the model [10].

section 4 on the facing page builds an application deployed on the cloud for collecting metrics. The methods described above are then applied on these metrics to determine the most relevant metrics for predicting the latency, given a partition size.

### 3.4. Applying the model for sizing and resource allocation

In this section the basic workflow of collecting metrics (training data) from applications running on the cloud is discussed. It is then shown how the proposed machine learning algorithm can be used to predict the performance of a sharding topology. The section also takes a brief look at predictive partitioning.

#### 3.4.1. Collecting metrics from applications running on the cloud

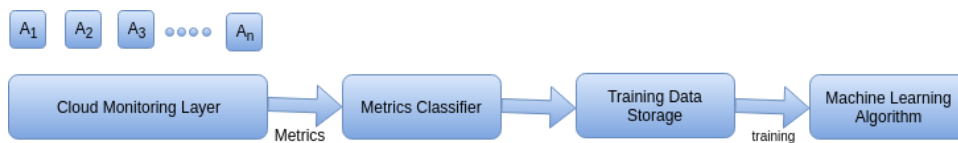


Figure 5: Using the prediction model for making predictions

Over its lifetime of operation, a cloud may host several applications. Many of these applications can be classified under a common type e.g. Payroll Management, Employee Management, Billing. Gusto, On-Pay, BambooHR, Xero etc are common examples of popular payroll management applications [25]

A machine learning algorithm will be able to use the information collected from the various deployments of a particular type of application to make accurate partitioning and sizing decisions for that application. The size of the training data increases as the number of instances of the application deployed increases. The increase in the size of the training data automatically increases the accuracy of the predictions made. Thus, a monitoring layer that classifies the metrics, collected from applications, into its different types will be able to build accurate prediction layers for each of them. This is shown in the fig. 5.

In the figure  $A_1, A_2, A_3 \dots A_n$  are applications running on the cloud. The cloud monitoring layer collects metrics from all the applications and hands them over to the metrics classifier. The metrics classifier organizes the metrics into different types. The classified metrics are used as the training data for the machine learning predictor, that predicts the performance of an application on a given cloud server configuration.

#### 3.4.2. Predicting the performance of a sharding topology

---

##### Algorithm 1 Verifying the shard server capacity

---

- 1)  $csm \leftarrow$  Cloud server metrics
  - 2)  $am \leftarrow$  Application metrics
  - 3)  $ds \leftarrow$  Size of data on the cloud server
  - 4)  $sla \leftarrow$  Expected latency of the application
  - 5)  $predicted\_latency \leftarrow ml\_predictor(csm, am, ds, sla)$
  - 6)  $capacity\_estimate \leftarrow (predicted\_latency - sla)$
  - 7) return  $capacity\_estimate$
-

The algorithm 1 accepts the cloud server metrics, the application metrics and the size of the data to be stored in the shard and returns the expected latency. It returns an integer `capacity_estimate` that can be interpreted as follows,

- `capacity_estimate < 0`: The predicted latency is lesser than the expected latency. A very large negative value implies that the server metrics have been over estimated and can be reduced.
- `capacity_estimate = 0`: The predicted latency is same as the expected latency. The server metrics have been accurately estimated and can be used to get the desired performance.
- `capacity_estimate > 0`: The predicted latency is greater than the expected latency. A very large positive value implies that the server capacity estimate is not enough and needs to be increased to get the desired performance.

The algorithm helps us determine if we should scale down or scale up to meet the required latency. This helps in the creation of accurate capacity estimations.

### **3.4.3. Performing predictive partitioning**

The machine learning algorithm proposed in this paper can be used together with a time series algorithm to predict the overload in a given shard. A time series algorithm can be used to determine the load (reads/writes) on a given shard and also the size of the shard, at a future time. This can be done by applying the algorithm on samples of the load and size of data at different times. The predicted load and the size of data in the shard can be used to estimate the latency of requests in the shard at a future time. Periodically applying the time series algorithm can help determine if the given setup will result in a latency for requests that is much larger than the expected latency for good performance, thus predicting overload. This is however out of scope for the current paper.

## **4. Quantitative Analysis**

The methods presented in this paper can be applied to a specific deployment, for multiple deployments from the same organization or at the cloud provider level itself.

When the methods collected are applied to a specific deployment, the collected metrics become very specific to that deployment. The data generated in this case would become available only after empirical analysis is performed on an application deployment in the cloud setup.

Applying these methods at the level of the cloud provider would need a more general collection of metrics. It will also help to classify the metrics specific to different applications; for example, the metrics collected for a payroll system would be very different from a system used for facial recognition. However, as the number of applications deployed on the cloud increases, it would help create recommendation systems which can be used to build accurate deployment templates depending on the nature of the application.

This section presents the empirical data collected for a specific application deployment and the analysis that was done on the collected data.

### **4.1. Application Setup and Data Generation**



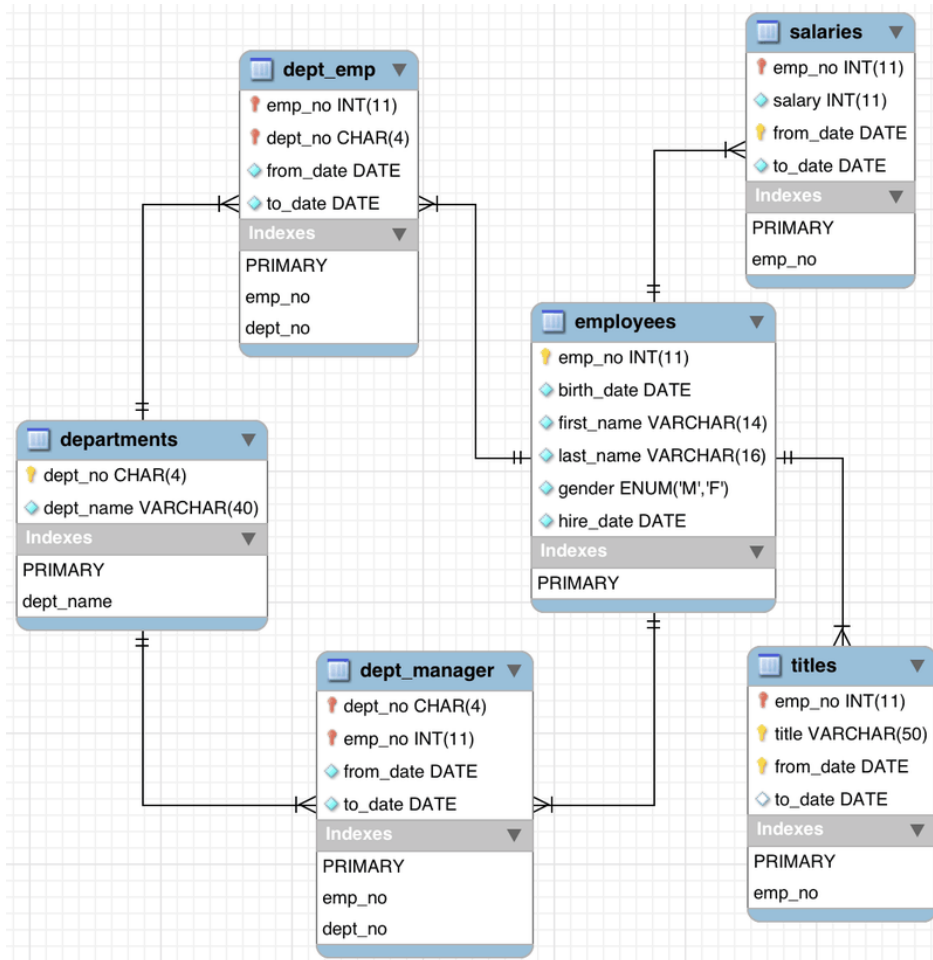


Figure 6: MySQL Employee Schema

The quantitative analysis was done using a standard employee database schema [14]. The employee database was sharded using the emp\_no as the sharding key. The shards were distributed on cloud servers obtained from cloudatcost [3]. Queries were run on this sharded setup for obtaining empirical metrics. Regression analysis was performed over these empirical metrics for building a machine learning model and for using it to make predictions. The following sections describe the setup used for data generation and follow it up with a detailed description of how the machine learning regression model is built and the measure of its validity.

Listing 1: Table Scan Query

```
SELECT COUNT( * ) FROM (
  SELECT S.emp_no, E.first_name, E.last_name, MAX(S.salary)
  FROM Salaries AS S, Employees AS E,
  WHERE S.emp_no = E.emp_no
        AND S.emp_no < %(max)s
        AND S.emp_no > %(min)s
  GROUP BY S.emp_no
)
```

fig. 6 on the previous page contains the Entity Relationship diagram of the schema used to store the data that was sharded. The schema represents the popular Employees Sample Database [14], which helps provide a large base of data spread over six separate tables. This structure is simple and easy to visualize, while at the same time being comprehensive.

The schema was populated with 5494 first names and 88799 last names. A full join of these two generated 5494 X 88799 unique names. The emp\_no was changed to an auto-generated column to generate primary keys. For each employee a birth\_date and hire\_date were randomly generated. An INSERT TRIGGER was created on the employees table for generating the salary history of an employee starting from the joining date to the current date. Each employee was assigned a department from one of the 9 available departments.

The query in listing 1 was used to generate load and also for latency measurements. The query was designed to perform a full scan of the data in the shard. Since the number of tuples in the shard corresponds to the size of the data in the shard, the performance of the query was directly influenced by the size of data in the shard. This query was fired on multiple client threads and the latencies were measured across multiple iterations of each of these client threads.

## 4.2. Cloud Setup

Table 2: Cloud Setup

Server	Configuration
VPS 1	2 Virtual Server Cores 2GB RAM
VPS 2	4 Virtual Server Cores 4GB RAM

The setup used to collect empirical metrics was hosted on the cloud provider - cloudfare [3]. Virtual Private Servers (VPS) were used to deploy both the application and the database instance.

Two different configurations (for the VPS) were used for creating the setup. The configurations used are shown in table 2. The application was deployed on one VPS, while the database was deployed on the other.

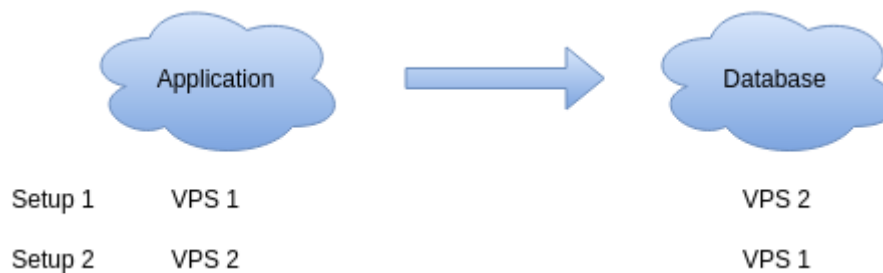


Figure 7: Cloud Setup

The two VPS were alternately used to deploy the application and the database, as seen in fig. 7. Data was collected from both these configurations. The advantages of doing so were twofold,

- Avoided bias introduced by a configuration.
- Ensured that the data collected represented the relationship between the parameters accurately.

## 4.3. Data Size and Metrics

The collected metrics helped establish the correlation between the shard size on a given setup and the latency. In order to create accurate predictions, a large volume of data needs to be collected. However, it would become difficult to present the entire set within this paper. Hence, this paper presents a reduced subset of the data size and metrics.

The following metrics were chosen for performing the regression analysis.

**Latency** The latency of a full table scan is indicative of the performance of the application. Predicting the latency of a table scan query given the data size and cloud machine configuration helps in starting with an ideal cloud topology, given the performance requirements of the application.

**Tuple Count** The tuple count is directly proportional to the size of the data handled by the query.

**innodb\_buffer\_pool\_size** The `innodb_buffer_pool_size` [7] indicates the amount of memory used by the MySQL storage engine to cache the table and the index data. A large value for this reduces the amount of disk I/O required to access the same relation data more than once.

Hence the `innodb_buffer_pool_size` configuration parameter was picked due to its correlation with the RAM size of the cloud server on which the MySQL Server is deployed.

#### 4.4. Data Collected

The goal of using multiple client threads to run the query was to create a real-time simulation of load. However, its performance is impacted due to contention with the other client threads. Thus, for a given configuration the spikes introduced due to the contention between the threads need to be evened out before performing regression analysis on the data.

Table 3: Data Set

Data Size (Tuple Count)	innodb_buffer_pool_size (MB)	Latency (ms)
100000	64	1467.5
200000	64	3209.5
300000	64	5723.35
400000	64	7926.8
500000	64	10415.05
100000	96	1498.75
200000	96	3109.5
300000	96	5777.5
400000	96	7902.5
500000	96	12672.5
100000	128	1341.25
200000	128	2769.75
300000	128	4469
400000	128	6094.75
500000	128	8103.75
100000	256	1317.5
200000	256	2645.5
300000	256	4348.5
400000	256	5866
500000	256	7635

Forty data points were collected on each thread for each combination of `innodb_buffer_pool_size` and data size. Four hundred data points were collected for each combination of `innodb_buffer_pool_size` and data size. The medians of these points were used to fix the latency for the `innodb_buffer_pool_size` and data size combination. A subset of this data is shown in the table 3.

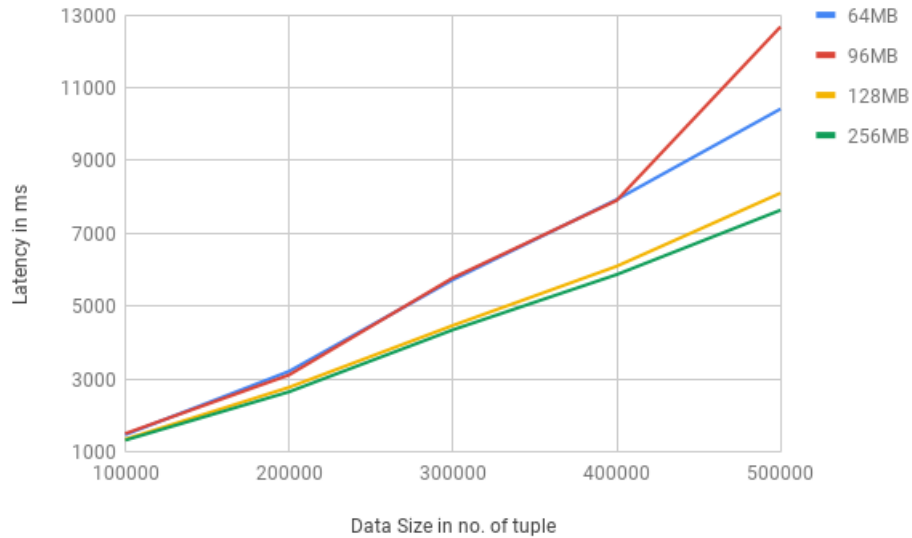


Figure 8: Latency vs Data Size

Each of the chosen metrics has a linear relation with the observed latency. This is shown in the graphs in fig. 8 and fig. 9. The graph in fig. 8 shows the relationship between the data size and the latency. As the size of the data increases the latency of the query increases, as expected.

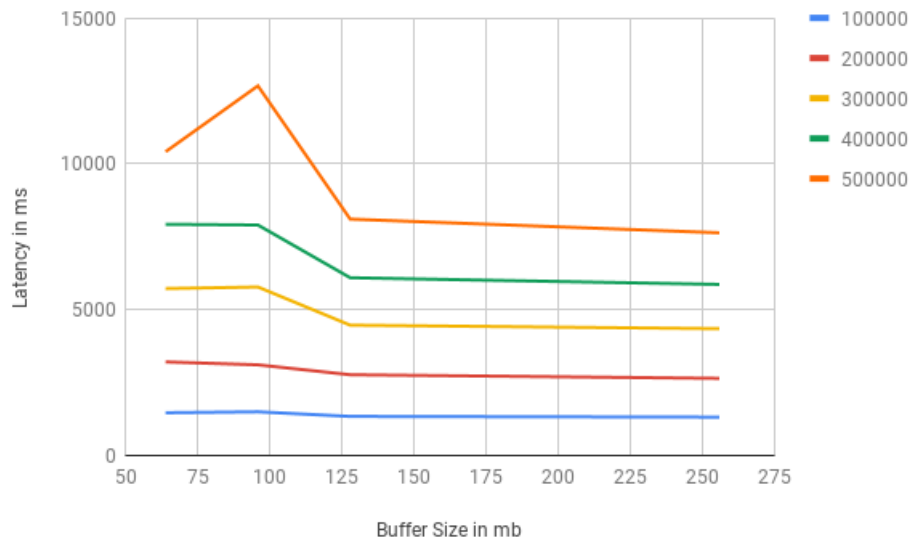


Figure 9: Latency vs InnoDB Buffer Pool Size

The graph in fig. 9 shows the relationship between the InnoDB buffer pool size and the latency. As the buffer size increases the latency drops, as expected. However, as the buffer size crosses 125 MB the latency benefit becomes less evident. 125 MB represents the threshold limit for the InnoDB buffer pool size for significant performance benefit given the available data size.

#### 4.5. Building the Prediction Model

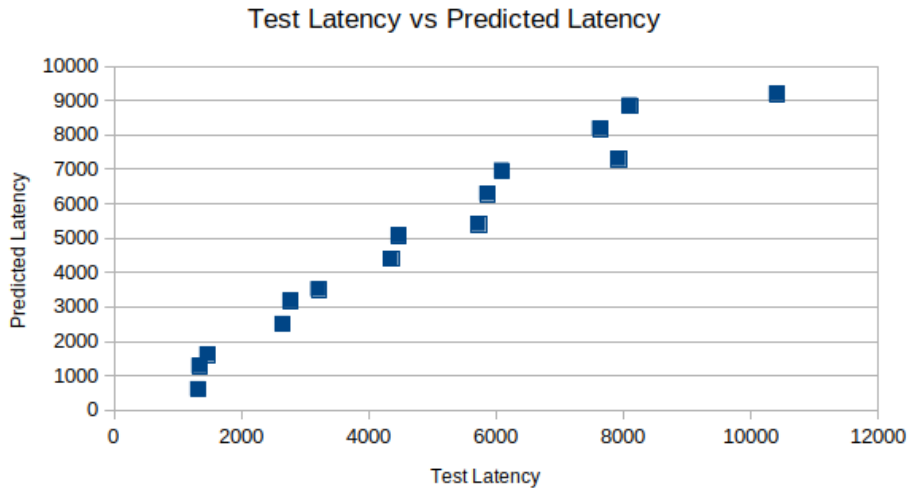


Figure 10: Test Latency vs Predicted Latency

The median of the latency observed on each of the threads and the overall median across threads on each configuration were used to create the data set. The K-Fold cross validation technique was used to form the training and the test sets. table 4 illustrates the test data vs the predicted latencies in milliseconds obtained by using the 5-Fold cross validation technique for 15 latencies from the collected data. The scatter graph for the same is presented in fig. 10.

Table 4: Test Latency vs Predicted Latency

Test Latency (ms)	Predicted Latency (ms)
1467.5	1616.55
3209.5	3511.31
5723.35	5406.06
7926.8	7300.82
10415.05	9195.58
1341.25	1279.75
2769.75	3174.51
4469	5069.26
6094.75	6964.02
8103.75	8858.78
1317.5	606.15
2645.5	2500.91
4348.5	4395.66
5866	6290.42
7635	8185.17

#### 4.6. Parameter Validation

##### 4.6.1. P – Value

Table 5: P-Value

SNo	Parameter	Value
1	innodb_buffer_pool_size	0.003
2	data_size	0.000

The p-value for the parameters in table 3 was calculated and it was found to be less than 0.05. This showed that there is a linear relationship between the dependent and the independent variables at a 5% level of

significance. This also validates our use of the multiple linear regression model for making predictions. The calculated p-values are shown in table 5.

#### 4.6.2. Adjusted R-squared

The adjusted R-squared for the data set was found to be 0.988. This value is closer to 10. When the `innodb_buffer_pool_size` was removed, the value was 0.978. When `data_size` was removed the value was found to be 0.496. Since removing either one of the parameters results in reducing the adjusted R-squared value, it can be established that the parameters improve the model more than might be attributable to chance.

#### 4.6.3. Variance Inflation Factors (VIFs)

Table 6: Variance Inflation Factors (VIFs)

SNo	Parameter	Value
1	<code>innodb_buffer_pool_size</code>	1
2	<code>data_size</code>	1

The variance inflation factors for both the parameters were calculated and were found to be lesser than 5. These VIFs are shown in the table 6. This shows that the variables do not contain redundant information and it also confirms the absence of multicollinearity.

### 5. Conclusion and future work

This paper showed how machine learning can be used to make accurate resource planning and sizing decisions. An application was created and deployed on the cloud. Empirical analysis was performed on this application for collecting metrics. Regression analysis was performed on the collected metrics for predicting the latency of the application on a given setup. P-Value, Adjusted R-squared and Variance inflation factors were used to establish the validity of the regression analysis. The predicted values were plotted against the actual values to verify the accuracy of the proposed methods.

A first step to extending the work in this paper would be to test the ideas on a more generic deployment at the organization level and then move up to the level of a cloud provider.

The next step would be to analyze the growth of the data on the shards and predict when it will hit the threshold predicted. Time series algorithms present one way of analyzing the data growth in the shards. Once it is possible to predict when the data size will hit the threshold, it should be possible to initiate the refinement proactively, rather than reactively.

The methods presented in this paper merely scratch the surface of the world of possibilities that surrounds the building of intelligent clouds. In addition to the obvious reduction in cost and efficient utilization these methods achieve, they set the platform for building intelligent recommendation systems for creating self-managed clouds.

#### Acknowledgments





I would like to express my deepest gratitude to Dr Suvamoy Changder and Dr Narayan C. Debnath without whose guidance and encouragement, the paper would not have taken shape.

#### References

- [1] Lionel C. Briand et al. "An Assessment and Comparison of Common Software Cost Estimation Modeling Techniques". In: Proceedings of the 21<sup>st</sup> International Conference on Software Engineering. ICSE '99. Los Angeles, California, USA: ACM, 1999, pp. 313–322.

- 
- [2] Cassandra Architecture. <https://docs.datastax.com/en/archived/cassandra/2.0/>. Accessed: Jan, 2019.
- [3] Cloud At Cost. <http://www.cloudatcost.com/>. Accessed: October, 2018.
- [4] Carlo Curino et al. "Schism: A Workload-driven Approach to Database Replication and Partitioning". In: Proc. VLDB Endow. 3.1-2 (Sept. 2010), pp. 48–57.
- [5] Deniz Hastorun et al. "Dynamo: amazon's highly available key-value store". In: In Proc. SOSP. 2007, pp. 205–220.
- [6] Chao-Wen Huang et al. "The improvement of auto-scaling mechanism for distributed database - A case study for MongoDB". In: Network Operations and Management Symposium (APNOMS), 2013 15th Asia Pacific. Sept. 2013, pp. 1–3.
- [7] InnoDB Buffer Pool Size. <https://dev.mysql.com/doc/refman/8.0/en/innodb-parameters.html>. Accessed: January, 2019.
- [8] S. Jamil et al. "Impact of facebook intensity on academic grades of private university students". In: 2013 5th International Conference on Information and Communication Technologies. Dec. 2013, pp. 1–10.
- [9] David Karger et al. "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web". In: In ACM Symposium on Theory of Computing. 1997, pp. 654–663.
- [10] Peter Kennedy. A Guide to Econometrics, 5th Edition. 5th ed. Vol. 1. The MIT Press, 2003.
- [11] P. Kookarinrat and Y. Temtanapat. "Analysis of Range-Based Key Properties for Sharded Cluster of MongoDB". In: Information Science and Security (ICISS), 2015 2nd International Conference on. Dec. 2015, pp. 1–4.
- [12] John J. Marciniak. Encyclopedia of Software Engineering. 2nd. New York, NY, USA: John Wiley & Sons, Inc., 2002. isbn: 0471210072.
- [13] Floyd A. Miller. "Improving Heuristic Regression Analysis". In: Proceedings of the 6<sup>th</sup> Annual Southeastern Regional Meeting of the Association for Computing Machinery and National Meeting of Biomedical Computing- Volume 1. ACM-SE 6. Chapel Hill, North Carolina: ACM, 1967, pp. 1–23.
- [14] MySQL Employee Sample Database. <https://dev.mysql.com/doc/employee/en/sakila-structure.html>. Accessed: January, 2019.
- [15] Sam Newman. Building Microservices. O'Reilly Media, Inc., Feb. 2015.
- [16] Oracle MySQL Cloud Service. <https://www.mysql.com/cloud/>. Accessed: 2018-06-22.
- [17] M. G. E. Peterson. "Multiple comparisons and the p-value in evaluation". In: Proceedings 12th IEEE Symposium on Computer-Based Medical Systems (Cat. No.99CB36365). 1999, pp. 260–263.
- [18] Man Qi et al. "Big Data Management in Digital Forensics". In: Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on. Dec. 2014, pp. 238–243.
- [19] Riak Architecture. <https://docs.basho.com/riak/kv/2.2.3/using/reference/v3-multi-datacenter/architecture/>. Accessed: January, 2019.
- [20] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321. Apr. 1992.
- [21] T. Rögnvaldsson et al. "Estimating p-Values for Deviation Detection". In: 2014 IEEE Eighth International Conference on Self-Adaptive and Self Organizing Systems. Sept. 2014, pp. 100–109.
- [22] Rebecca Taft et al. "E-store: Fine-grained Elastic Partitioning for Distributed Transaction Processing Systems". In: Proc. VLDB Endow. 8.3 (Nov. 2014), pp. 245–256.2735514.
- [23] Rebecca Taft et al. "P-Store: An Elastic Database System with Predictive Provisioning". In: Proceedings of the 2018 International Conference on Management of Data. SIGMOD '18. Houston, TX, USA: ACM, 2018, pp. 205–219. isbn: 978-1-4503-4703-7.
- [24] Hee Beng Kuan Tan, Yuan Zhao, and Hongyu Zhang. "Conceptual Data Model-based Software Size Estimation for Information Systems". In: ACM Trans. Softw. Eng. Methodol. 19.2 (Oct. 2009), 4:1–4:37
- [25] Typical cloud applications. <https://financesonline.com/top-15-payroll-management-software-systems/>. Accessed: October, 2018.
- [26] Xiaolin Wang, Haopeng Chen, and Zhenhua Wang. "Research on Improvement of Dynamic Load Balancing in MongoDB". In: Dependable, Autonomic and Secure Computing (DASC), 2013 IEEE 11<sup>th</sup> International Conference on. Dec. 2013, pp. 124–130.
- [27] Wikipedia page view statistics. <https://dumps.wikimedia.org/other/pageviews/2018/>. Accessed: January, 2019.
-

## Authors

	<p><b>Narayanan Venkateswaran</b> is a researcher in the Computer Science and Engineering Department of National Institute of Technology Durgapur. He has over 13 years of experience in the industry having worked in Enterprise DB, Oracle, Amazon, Microsoft and Sun Microsystems. He is an open source committer in the Apache Derby project and has contributed extensively to several databases, including JavaDB and MySQL.</p>
	<p><b>Anurag Shekhar</b> is a Senior Principal Member Technical Staff at Oracle India Pvt Ltd. He has over 22 years of experience in the Software industry, having worked in Sun Microsystems, IBM and Oracle. He has contributed heavily to several databases including, DB2, MySQL and JavaDB. He has mentored several junior engineers and has worked extensively in the area of database internals and file systems.</p>
	<p><b>Dr. Suvamoy Changder</b> is an Assistant professor in the Department of Computer Science and Engineering of National Institute of Technology Durgapur. He completed his PhD in Information Security, with specialization in Steganography and watermarking, from NIT Durgapur. He has over 15 years of teaching experience in addition to several years in the information technology industry before that. Some of the subjects that Dr. Suvamoy handles include Data Structures, Design and Analysis of Algorithms, Information and Coding Theory, Database Management Systems.</p>
	<p><b>Dr. Narayan C. Debnath</b> has been a Full Professor of Computer Science since 1989 and currently the Chairman of Computer Science at Department of Software Engineering, Eastern International University, Vietnam. He is also serving as the Director and Past-President of the International Society for Computers and Their Applications (ISCA). Dr. Debnath is a recipient of a Doctorate degree in Computer Science and a Doctorate degree in Applied Physics (Electrical Engineering). In the past, he served as the President, Vice President, and Conference Coordinator of the International Society for Computers and Their Applications (ISCA), and has been a member of the ISCA Board of Directors since 2001. He served as the Acting Chairman of the Department of Computer Science at Winona State University and received numerous Honors and Awards. During 1986-1989, Dr. Debnath was a faculty of Computer Science at the University of Wisconsin-River Falls, USA, where he was nominated for the National Science Foundation Presidential Young Investigator Award in 1989.</p>