# Module 5
# Machine Learning Algorithms for Big Data Analytics

## *5.1 Introduction*

***Artificial Intelligence*** **(AI)** refers to the science and engineering of making computers perform tasks, which normally require human intelligence. For example, tasks such as predicting future results, visual perception, speech recognition, decision making and natural language processing.

Two concepts in AI, 'machine learning' and 'deep learning' provide powerful tools for advanced analytics and predictions.

### Machine Learning

Machine Learning (ML) is a field of computer science based on AI which deals with learning from data in three phases, i.e. collect, analyze and predict. It does not rely on explicitly programmed instructions.

An ML program learns the behavior of a process. The program uses data generated from various sources for training. Learning from the outcomes from common inputs improves future performance from previous outcomes. Learning applies in many fields of research and industry. Learning from study of data enables efficient and logical decisions for future actions.

***Deep Learning*** **(DL)** refers to structured learning (DSL) or hierarchical learning. DL methods are advanced methods, such as artificial neural networks (ANN) such as artificial neural networks (ANN) or neural nets, deep neural networks, deep belief networks and recurrent neural networks. Learning can be unsupervised, semi-supervised or supervised. Applications of DL and ANN include computer vision, speech recognition, Natural Language Processing (NLP), audio recognition, social network filtering, machine translation, bioinformatics and drug design. DL methods give results comparable to and in some cases superior to human experts.

## 5.2 ESTIMATING THE RELATIONSHIPS, OUTLIERS, VARIANCES, PROBABILITY DISTRIBUTIONS AND *CORRELATIONS*

Independent variables represent directly measurable characteristics. For example, year of sales figure or semester of study. Dependent variables represent the characteristics. For example, profit during successive years or grades awarded in successive semesters. Values of a dependent variable depend on the value of the independent Variable.

***Predictor variable*** is an independent variable, which computes a dependent variable using some equation, function or graph, and does a prediction. For example, predicts sales growth of a car model after five years from given input datasets for the sales, or predicts sentiments about higher sales of particular category of toys next year.

*Outcome variable* represents the effect of manipulation(s) using a function, equation or experiment. For example, CGPA (Cumulative Grade Points Average) of the student or share of profit to each shareholder in a year using profit as the dependent variable. CGPA of a student computes from the grades awarded in the semesters for which student completes his/her studies. A company declares the share of profit to each shareholder in a year after subtracting requirements of money for future growth from the profit.

*Explanatory variable* is an independent variable, which explains the behavior of the dependent variable, such as linearity coefficient, non-linear parameters or probabilistic distribution of profit-growth as a function of additional investment in successive years.

*Response variable* is a dependent variable on which a study, experiment or computation focuses. For example, improvement in profits over the years from the investments made in successive years or improvement in class performance is measured from the extra teaching efforts on individual students of a class.

*Feature variable* is a variable representing a characteristic. For example, apple feature red, pink, maroon, yellowish, yellowish green and green. Feature variables are generally represented by text characters. Numbers can also represent features. For example, red with 1, orange with 2, yellow with 3, yellowish green 4 and green 5.

*Categorical variable* is a variable representing a category. For example, car, tractor and truck belong to the same category, i.e., a four-wheeler automobile. Categorical variables are generally represented by text characters.

## Relationships-Using Graphs, Scatter Plots and Charts

A relationship between two or more quantitative dependent variables with respect to an independent variable can be well-depicted using graph, scatter plot or chart with data points, shown in distinct shapes. Conventionally, independent variables are on the x-axis, whereas the dependent variables on the y-axis in a graph. A line graph uses a line on an x-y axis to plot a continuous function.

A scatter plot is a plot in which dots or distinct shapes represent values of the dependent variable at the multiple values of the independent variable Whether two variables are related to each other or not, can be derived from statistical analysis using scatter plots.

## Linear and Non-linear Relationships

A linear relationship exists between two variables, say x and y, when a straight line (y = a0 + a1.x) can fit on a graph, with at least some reasonable degree of accuracy. The a1 is the linearity coefficient. For example, a scatter chart can suggest a linear relationship, which means a straight line. Figure 6.1 shows a scatter plot, which fits a linear relationship between the number of students opting for computer courses in years between 2000 and 2017.

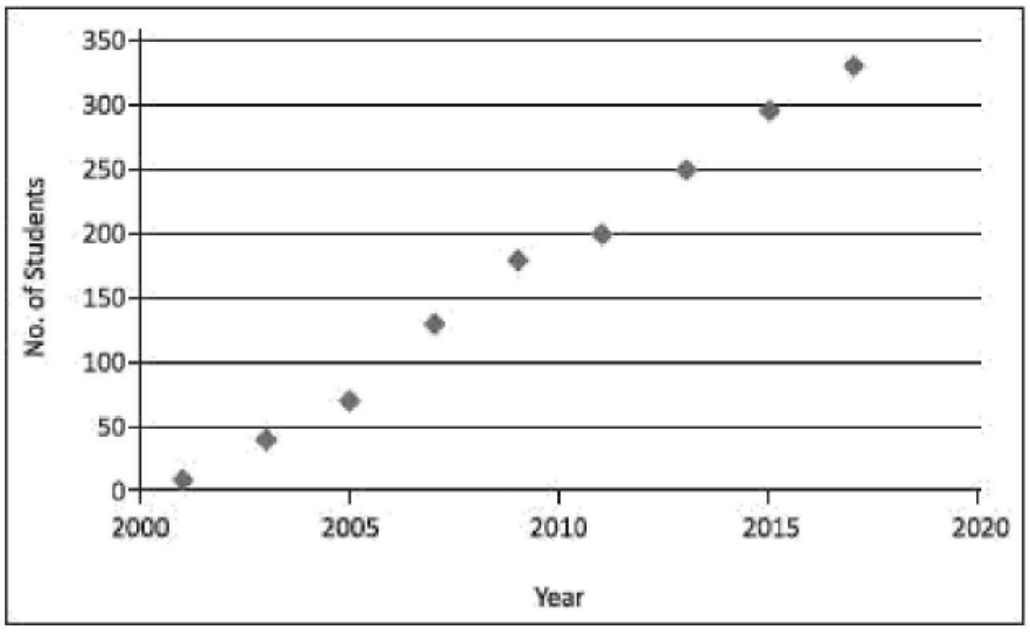



**Figure 5.1 Scatter plot for linear relationship between students opting for computer courses in years between 2000 and 2017**

A linear relationship can be positive or negative. A positive relationship implies if one

variable increases in value, the other also increases in value. A negative relationship, on the other hand, implies when one increases in value, the other decreases in value. Perfect, strong or weak linearship categories depend upon the bonding between the two variables.

A non-linear relationship is said to exist between two quantitative variables when a curve (y = a0 + a1.x + a2.x2 + •••) can be used to fit the data points. The fit should be with at least some reasonable degree of accuracy for the fitted parameters, a0, a1, a2 ••• Expression for y then generally predicts the values of one quantitative variable from the values of the other quantitative variable with considerably more accuracy than a straight line.

Consider an example of non-linear relationship: The side of a square and its area are not linear. In fact, they have quadratic relationship. If the side of a square doubles, then its area increases four times. The relationship predicts the area from the side.

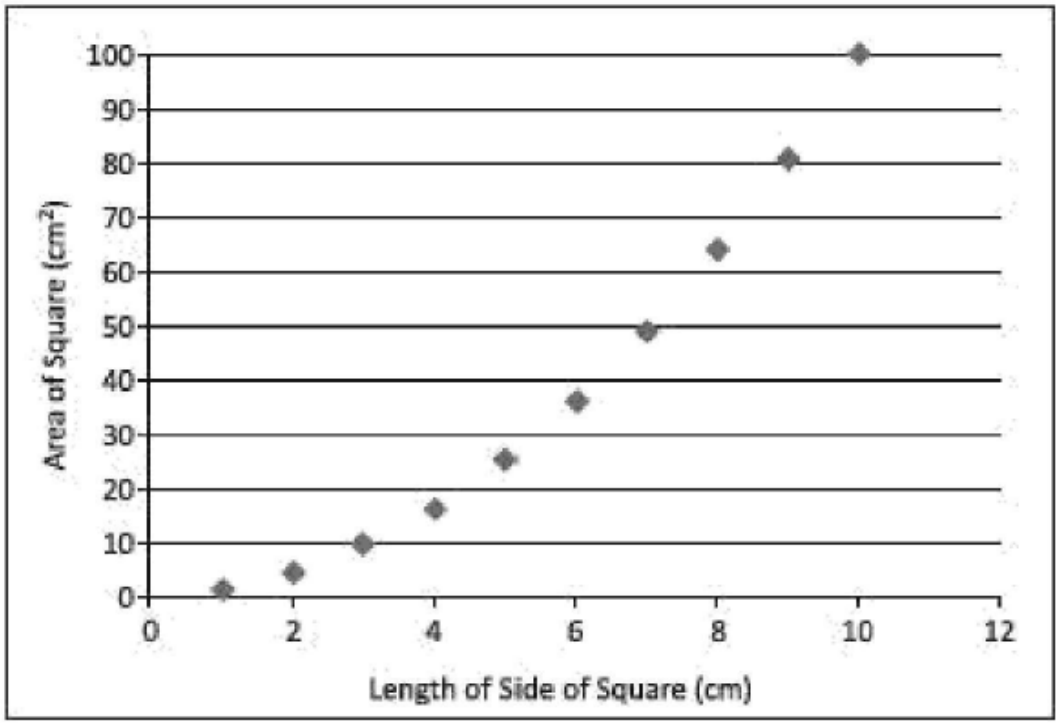



**Figure 5.2 scatter plot in case of a non-linear relationship between side of square and its area**

## Estimating Relationship

Estimating the relationships means finding a mathematical expression, which gives the value of the variable according to its relationship with other variables. For example, assume Ym = sales of a car model m in xth year of the start of manufacturing that model.

## Outliers

Outliers are data points that are numerically far distant from the rest of the points in a dataset, are termed as outliers. Outliers show significant variations from the rest of the points. Identification of outliers is important to improve data quality or to detect an anomaly

There are several reasons for the presence of outliers in relationships. Some of these are:

- Anomalous situation
- Presence of a previously unknown fact
- Human error (errors due to data entry or data collection)
- Participants intentionally reporting incorrect data (This is common in self-reported measures and measures that involve sensitive data which participant doesn't want to disclose)
- Sampling error (when an unfitted sample is collected from population).

Population means any group of data, which includes all the data of interest. For example, when analyzing 1000 students who gave an examination in a computer course, then the population is 1000. 100 games of chess will represent the population in analysis of 100 games of chess of a grandmaster.

Sample means a subset of the population. Sample represents the population for uses, such as analysis and consists of randomly selected data.

## Variance

Variance measures by the sum of squares of the difference in values of a variable with respect to the expected value. Variance can alternatively be a sum of squares of the difference with respect to value at an origin. Variance indicates how widely data points in a dataset vary. If data points vary greatly from the mean value in a dataset, the variance is large; otherwise, the variance is less. The variance is also a measure of dispersion with respect to the expected value.

A high variance indicates that the data in the dataset is very much spread out over a large area (random dataset), whereas a low variance indicates that the data is very similar in nature.

No variance is sometimes hard to understand in real datasets

## *Standard Deviation and Standard Error Estimates*

**Standard Deviation** With the help of variance, one can find out the standard deviation. Standard deviation, denoted by s, is the square root of the variance. The s says, "On an average how far do the data points fall from the mean or expected outcome?" Though the interpretation is the same as variance buts is squared rooted, therefore, less susceptible to the presence of outliers. The formulae for the population and the sample standard deviations are as follows:

$$\text{The Population Standard Deviation: } \sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i-\mu)^2}$$

$$\text{The Sample Standard Deviation: } \sigma \approx \sqrt{\frac{1}{S-1}\sum_{i=1}^{S}(x_i-\bar{x})^2},$$

where N is number of data points in population, S is number in the sample, m is expected in the population or average value of x, and x is expected x in the sample.

**Standard Error** The standard error estimate is a measure of the accuracy of predictions from a relationship. Assume the linear relationship in a scatter plot of y (Figure 6.1). The scatter plot line, which fits, is defined as the line that minimizes the sum of squared deviations of prediction (also called the sum of squares error). The standard error of the estimate is closely related to this quantity and is defined below:

$$\sigma_{est} = \sqrt{\frac{\sum(y-y')^2}{N}},$$

where sest is the standard error in the estimate, y is an observed value, y¢ is a predicted value, and N is the number of values observed. The standard error estimate is a measure of the dispersion (or variability) in the predicted values from the expression for relationship.

### Probabilistic Distribution of Variables, Items or Entities

Probability is the chance of observing a dependent variable value with respect to some independent variable. Suppose a Grandmaster in chess has won 22 out of 100 games, drawn 78 times, and lost none. Then, probability P of winning Pw is 0.22, P of drawn game P0 is 0.78 and P of losing, PL = 0. The sum of the probabilities is normalized to 1, as only one of the three possibilities exist.

Probability distribution is the distribution of P values as a function of all possible independent values, variables, situations, distances or variables. For example, if Pis given by a function P(x), then P varies as x changes. Variations in P(x) with x can be discrete or continuous. The values of P are normalized such that sum of all P values is 1. Assuming distribution is around the expected value x, the standard normal distribution formula is:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\bar{x})^2}{2\sigma^2}}$$

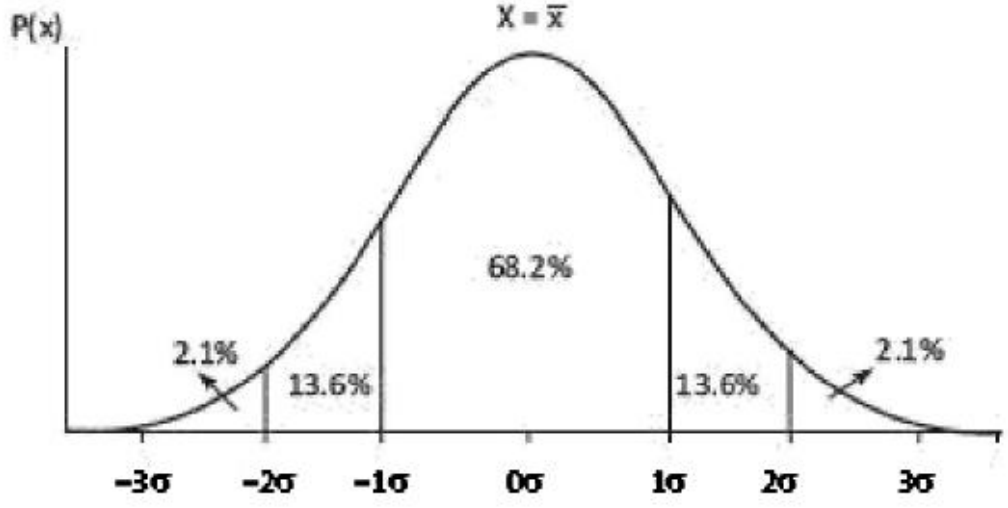



**Figure 5.3 Probability distribution function as a function of *x* assuming normal distribution around *x* = 'i, and standard deviation = s**

The figure also shows the percentages of areas in five regions with respect to the total area under the curve for P(x). The variance for probability distribution represents how individual data points relate to each other within a dataset.

variance is the average of the squared differences between each data value and the mean.

**Kernel Functions**

Kernel function is a function which is a central or key part of another function. For example, Gaussian kernel function is the key part of the probability distribution function. Figure 5.3 shows the probability normal distribution, which is a Gaussian function based on the Gaussian kernel function.

A kernel function[1], K* defines as

$$K^*(u) = \lambda . K(\lambda . u),$$

where $\lambda > 0$. Gaussian kernel function is

$$K^*(x) = \left[\frac{1}{(\sqrt{2\pi})}\right] e^{\left[-\frac{u^2}{2^2}\right]},$$

and when $u = \frac{\left\{x - \bar{x}\right\}}{\sigma}$, the distribution function is proportional to $P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\bar{x})^2}{2\sigma^2}}$.

$$\lambda = \left\{\frac{1}{\sigma\sqrt{2}}\right\}$$

Tricube kernel function is:

$$K^*(u) = (70/81)\ (1 - |u|^3)^3 \lambda . K(\lambda . u),$$

where $|u| \leq 1$.

## Moments

*Moments* (0, 1, 2, ...) refer to expected values to the powers of (0, 1, 2 ...) of random variable variance. $0^{th}$ moment is 1, $1^{st}$ moment = $E(x) = \bar{x}$, (expected value), $2^{nd}$ moment is squared $V[(x_i - \bar{x},)^2]$ = sum of product of $(x_i - \bar{x},)^2$, and $P(x = x_i)$.

## Analysis of Variance

An ANOVA test is a method which finds whether the fitted results are significant or not. This means that the test finds out (infer) whether to reject or accept the null hypothesis. Null hypothesis is a statistical test that means the hypothesis that "no significant difference exists between the specified populations difference is just due to sampling or experimental error.

Consider two specified populations (datasets) consisting of yearly sales data of Tata Zest andJaguar Land Rover models. The statistical test is for proving that yearly sales of both the models, means increments and decrements of sales are related or not. Null hypothesis starts with the assumption that no significant relation exists in the two sets of data (population).

The analysis (ANOVA) is for disproving or accepting the null hypothesis. The test also finds whether to accept another alternate hypothesis. The test finds that whether testing groups have any difference between them or not.

F-test F-test requires two estimates of population variance- one based on variance between the samples and the other based on variance within the samples. These two estimates are then compared for F-test:

$$F = \frac{El(V)}{E2(V)}$$

where El(V) is an estimate of population variance between the two samples and E2(V) is an estimate of population variance within the two samples. Several different F-tables exist. Each one has a different level of significance. Thus, look up the numerator degrees of freedom and the denominator degrees of freedom to find the critical value.

## Correlation

Correlation means analysis which lets us find the association or the absence of the relationship between two variables, x and y. Correlation gives the strength of the relationship between the model and the dependent variable on a convenient 0-100% scale.

R-Square Risa measure of correlation between the predicted values y and the observed values of x. R-squared (R2 ) is a goodness-of-fit measure in linear- regression model. It is also known as the coefficient of determination. R2 is the square of R, the coefficient of multiple correlations, and includes additional independent (explanatory) variables in regression equation.

Interpretation of R-squared The larger the R2 , the better the regression model fits the observations, i.e., the correlation is better. Theoretically, if a model shows 100% variance, then the fitted values are always equal to the observed values, and therefore, all the data points would fall on the fitted regression line

## Correlation Indicators of Linear Relationships

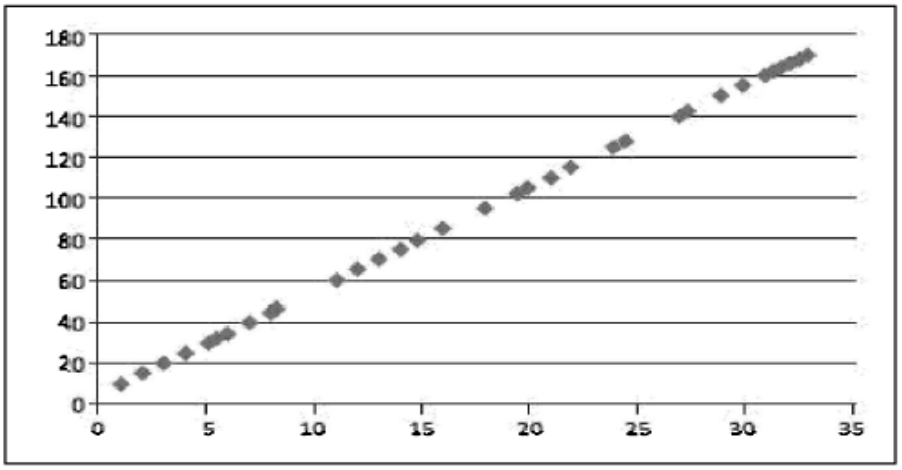
Correlation is a statistical technique that measures and describes the 'strength' and 'direction' of the relationship between two variables.

Relationships and correlations enable training model on sample data using statistical or ML algorithms. Statistical correlation is measured by the coefficient of correlation. The most common correlation coefficient, called the Pearson product-moment correlation coefficient. It measures the strength of the linear association between variables. The correlation r between the two variables x and y is:

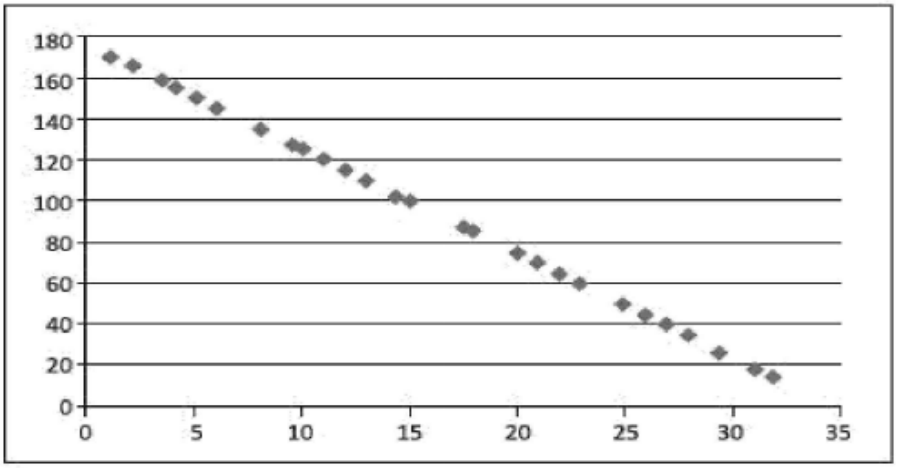
$$r = \left[\frac{1}{(n-1)}\right] \times \sum \left\{ \left[\frac{(x_i - \overline{x})}{\sigma_x}\right] \times \left[\frac{(y_i - \overline{y})}{\sigma_y}\right] \right\},$$

where n is the number of observations in the sample, xi is the x value for observation i, x- is the sample mean of x,Yi is the y value for observation i, y- is the sample mean of y, sx is the sample standard deviation of x, and Sy is the sample standard deviation of y.

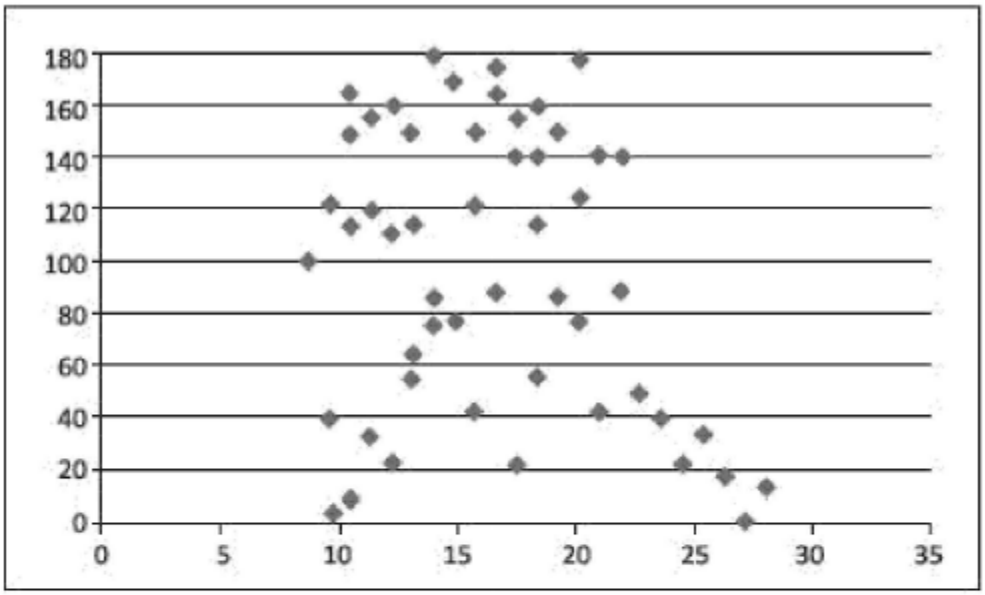
| Value of r | Strength of relationship |
|---|---|
| -1.0 to -0.5 or 1.0 to 0.5 | Strong |
| -0.5 to -0.3 or 0.3 to 0.5 | Moderate |
| -0.3 to -0.1 or 0.1 to 0.3 | Weak |
| -0.1 to 0.1 | None or very weak |



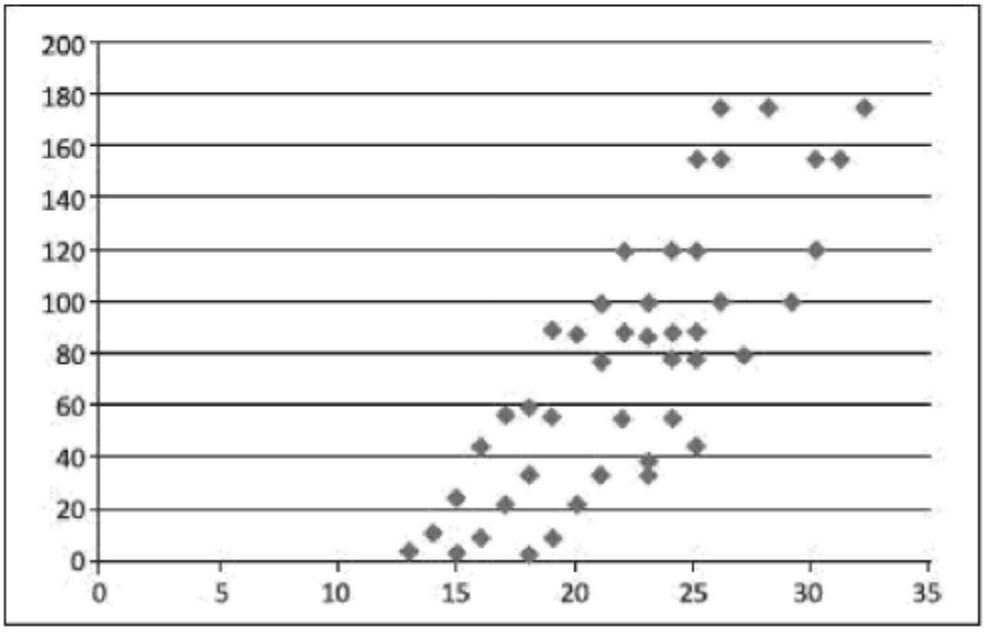

Perfect Positive Linear Relationship ($r = 1$)




Perfect Negative Linear Relationship ($r = -1$)




No Relationship ($r \sim 0$)




Positive Linear Relationship ($r = 0.9$)

**Figure 5.4 Perfect and imperfect, linear positive and negative relationships, and the strength and direction of the relationship between variables**

### 5.3 Regression Analysis

Correlation and regression are two analyses based on multivariate distribution. A multivariate distribution means a distribution in multiple variables.

Suppose a company wishes to plan the manufacturing of Jaguar cars for coming years. The company looks at sales data regressively, i.e., data of

previous years' sales. Regressive analysis means estimating relationships between variables. Regression analysis is a set of statistical steps, which estimate the relationships among variables. Regression analysis may require many techniques for modeling and performing the analysis using multiple variables. The aim of the analysis is to find the relationships between a dependent variable and one or more independent, outcome, predictor or response variables. Regression analysis facilitates prediction of future values of dependent variables

Non-linear regression equation is as follows:

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3,$$

where number of terms on the right-hand side are 3 or 4. Linear regression means only the first two terms are considered. The following subsections describe regression analysis in detail.

**Simple Linear Regression**

Linear regression is a simple and widely used algorithm. It is a supervised ML algorithm for predictive analysis. It models a relationship between the independent predictor or explanatory, and the dependent outcome or variable, y using a linearity equation.

$$y = f(a_0, a_1) = a_0 + a_1 x,$$

where $a_0$ is a constant and $a_1$ is the linearity coefficient.

Simple linear regression is performed when the requirement is prediction of values of one variable, with given values of another variable.

The purpose of regression analysis is to come up with an equation of a line that fits through a cluster of points with minimal amount of deviation from the line. The best-fitting line is called the regression line. The deviation of the points from the line is called an 'error'.

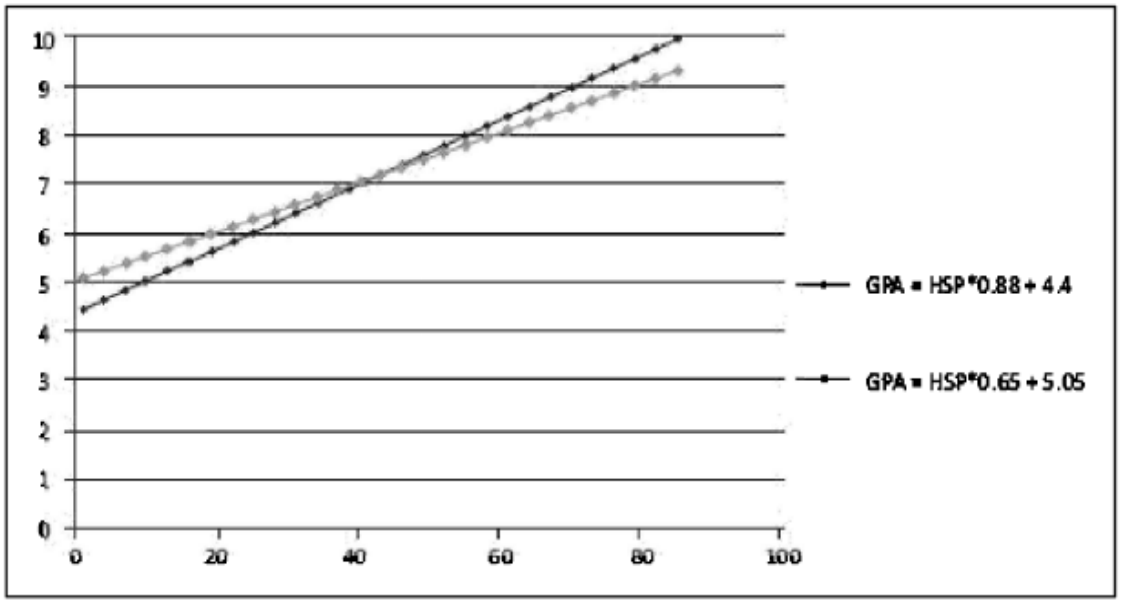



Figure shows a simple linear regression with two regression lines with different regression equations. Looking at the scatter plot, two lines can fit best to summarize the relation between GPA and high school percentage.

**Linear Square Estimation**

Assume n data-points, i = 1, 2, ..., n. A line out of two lines (Figure 6.6) that fits the data best will be one for which the sum of the squares of the n prediction errors (one for each observed data point) is as small as possible. This is the 'least squares criterion', which says that the best fit is one, which 'minimizes the sum of the squared prediction errors'. This implies that when the equation of the best fitting line is:

$$y_i' = b_0 + b_1 x_i$$

where b0 and b1 are the coefficients which minimize the errors. The coefficients values make the sum of the squared prediction errors as small as possible

**Multiple Regression**

A criterion variable can be predicted from one predictor variable in simple linear regression. The criterion can be predicted by two or more variables in multiple regressions.

Multiple regressions are used when two or more independent factors are involved. These regressions are also widely used to make short- to mid-term predictions to assess which factors to include and which to exclude. Multiple regressions can be used to develop alternate models

with different factors. More than one variable can be used as a predictor with multiple regressions. However, it is always suggested to use a few variables as predictors necessarily, to get a reasonably accurate forecast. The prediction takes the form:

$$y = a + c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

More than one variable can be used as a predictor with multiple regressions. However, it is always suggested to use a few variables as predictors necessarily, to get a reasonably accurate forecast. The prediction takes the form:

Multiple regression analysis, often referred to simply as regression analysis, examines the effects of multiple independent variables on the value of a dependent variable or outcome.

**Modelling Possibilities using Regression.**

Regressions range from simple models to highly complex equations. Two primary uses for regression are forecasting and optimization. Consider the following examples:

1. Using linear analysis on sales data with monthly sales, a company could forecast sales for future months.
2. For the funds that a company has invested in marketing a particular brand, an analysis of whether the investment has given substantial returns or not can be made.
3. Suppose two promotion campaigns are running on TV and Radio in parallel. A linear regression can confine the individual as well as the combined impact of running these advertisements together.
4. An insurance company exploits a linear regression model to obtain a tentative premium table using predicted claims to Insured Declared Value ratio.
5. A financial company may be interested in minimizing its risk portfolio and hence want to understand the top five factors or reasons for default by a customer.
6. To predict the characteristics of child based on the characteristics of their

parents.

7. A company faces an employment discrimination matter in which a claim that women are being discriminated against in terms of salary is raised.
8. Predicting the prices of houses, considering the locality and builder characteristics in a locality of a particular city.
9. Finding relationships between the structure and the biological activity of compounds through their physical, chemical and physicochemical traits is most commonly performed with regression techniques.
10. To predict compounds with higher bioactivity within groups.

### 6.3.2 Predictions using Regression Analysis

Regression analysis is a powerful technique used for predicting the unknown value of a variable from the known value of another variable. Regression analysis is generally a statistical method to deal with the formulation of a mathematical model depicting the relationship amongst dependent and independent variables. The dependent variable is used for the purpose of prediction of the values. One or more variables whose values are hypothesized are called independent variables. The prediction for the dependent variable can be made by accurate selection of independent variables to estimate a dependent variable.

Two steps for predicting the dependent variable:

Estimation step: A function is hypothesized and the parameters of the function are estimated from the data collected on the dependent variable.

Prediction step: The independent variable values are then input to the parameterized function to generate predictions for the dependent variable.

K-Nearest-Neighbour Regression Analysis

Consider the saying, 'a person is known by the company he/she keeps.' Can a prediction be made using neighbouring data points? K-Nearest Neighbours (KNN) analysis is an ML based technique using the concept, which uses a subset of K = 1, 2 or 3 neighbours in place of a complete dataset. The subset is a training dataset.

Assume that population (all data points of interest) consist of k-data points. A data point independent variable is xi, where i = 1 to k. K-Nearest Neighbours (KNN) is an algorithm,

which is usually used for classifiers. However, it is useful for regression also. Predictions can use all k examples (global examples) or just K examples (K-neighbours with K = 1, 2 or 3). It predicts the unknown value Yp using predictor variable using the available values at the neighbours. The training dataset consists of available values of Yni at Xni with ni = 1 to K, where ni is the K-the neighbour, means just the local examples.

A subset of training dataset restricts k to K-neighbours, where K = 1, 2 or 3. This means using local values near the predictor variable. K = 1 means the nearest neighbour data points. K = 2 means the next nearest neighbour data points (xi,YJ K = 3 means the next to next nearest neighbour data points (Xj,yi).

First find all available neighbouring target (xi, Yi) cases and then predict the numerical value to be predicted based on a similarity measure. Prediction methods are as follows:

1. Simple interpolation, when predictor variable is outside the training subset
2. Extrapolation, when predictor variable is outside the training subset
3. Averaging, local linear regression or local-weighted regression.

**Euclidean Distance** The following equation computes the Euclidean distance $D_{Eu}$:

Sum of the squared Euclidean distance, $[D_{Eu}]^2 = \left[\sum_{i=1}^{v}(x_i - x_i')^2\right]$, and

$$\text{Euclidean distance } D_{Eu} = \left[\sum_{i=1}^{v}(x_i - x_i')^2\right]^{1/2}$$

$$\text{Euclidean distance } D_{Eu} = [(x_j - x_{j+1})^2 + (y_j - y_{j+1})^2]^{1/2}$$

**Manhattan Distance** The following equation computes the Manhattan distance $D_{Ma}$:

Manhattan distance $D_{Ma}$ =

$$D_{Ma} = \sum_{i=1}^{v}\left[\left|x_i - x_i'\right|\right] \quad (6.20c)$$

Manhattan distance for three variables $v = 3$ (two independent variables and one dependent variable case) consists of three terms on the right-hand side in

Manhattan distance for three variables v = 3 (two independent variables and one dependent variable case) consists of three terms on the right-hand side in Equation.

**Comparison between Euclidean and Manhattan Distances**

Basically, Euclidean distance is the direct path distance between two data points in v-dimensional metric spaces. Manhattan distance is the staircase path distance between them. Staircase distance means to move to the next point, first move along one metric dimension (say, x axis) from the first point, and then move to the next along another dimension (say, y axis).

When v = 2, Euclidean distance is the diagonal distance between the points on an x-y graph. Manhattan distances are faster to calculate as compared to Euclidean distances. Manhattan distances are proportional to Euclidean distances in case of linear regression.

**Minkowski Distance** The following equation computes the Minkowski distance $D_{Mi}$:

$$\text{Minkowski distance } D_{Mi} = \left\{ \sum_{i=1}^{v} \left[ (x_i - x_i')^q \right] \right\}^{1/q}$$

**Hamming Distance** When predictions are on the basis of categorical variables, then use the Hamming distance. It is a measure of the number of instances in which corresponding values are found.

$$\text{Hamming Distance, } D_H = \sum_{i=1}^{v} |x_i - x_i'|,$$

when xi= x¢i then DH= O and when Xj not equal to x¢i then DH= 1. For example, Hamming distance DH= 1 between 10100111100 and 11100111100 because just one substitution is needed, change second bit from O to 1 at 10th place from the right to left positioned bits.

**Normalization Concept** Normalization factor in p-norm form in a v- dimensional space is

$$x_i = N^{-1} \cdot x_i \text{ where } N = \left(\sum_{i=1}^{v} |x_i|^p\right)^{1/p}$$

Here, xi is ith component of the vector X. The total number of components are Two-dimensional space v = 2, three-dimensional v = 3. The following example explains the meaning of distances, use of Euclidean and Manhattan distances, use distances for predictions, and the KNN regression analysis.

**5.4 FINDING SIMILAR ITEMS, SIMILARITY OF SETS ANDCOLLABORATIVE FILTERING**

The following subsections describe methods of finding similar items using similarities, application of near-neighbour search, Jaccard similarity of sets, similarity of documents, Collaborative Filtering (CF) as a similar-set problem, and the distance measures for finding similarities.

Finding Similar Items

An analysis requires many times to find similar items. For example, finding similar excellent performance of students in Python programming, similar showrooms of a specific car model which show high sales per month, recommending books on similar topic such as in Internet of Things by Raj Kamal from McGraw-Hill Higher Education, etc.

Application of Near Neighbour Search

Similar items can be found using Nearest Neighbour Search (NNS). The search finds that a point in a given set is most similar (closest) to a given point. A dissimilarity function having larger value means less similar. The dissimilarity function is used to find similar items.

NNS algorithm is as follows: Consider set S having points in a space M. Consider a queried point q EM, which means q is member of M. k-NNS algorithm finds the k-closet (1-NN) points to q in S.

Do not consider the number of items in which two users' preferences overlap. (e.g., 2 overlap items==> 1, more items may not be better.)

If two users overlap on only one item, no correlation can be computed.

The correlation is undefined if series of preference values are identical.

Greater distance means greater dissimilarity. Dissimilarity coefficient relates to a distance metric in metrics space in v-dimensional space. An algorithm computes Euclidean, Manhattan and Minkowski distances using Equations.

Distance metric is symmetric and follows triangular inequality. Meaning of triangular inequality can be understood by an example. Consider three vectors of lengths x, y, and z. Then, triangular inequality means

$z < x + y$. It is similar to the theorem of inequality that the third side of a triangle is less than the sum of two other sides, and never equal. The theorem applies to v-dimensional space also. Dissimilarity can be asymmetric, i.e., triangular inequality is not true (Bergman divergence).

**Jaccard Similarities set**

Let A and B be two sets. Jaccard similarity coefficient of two sets measures using notations in set theory as shown below:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

A n B means the number of elements or items that are same in sets A and B. A U B means the number of elements or items present in union of both the sets. Assume two set of students in two computer courses, Computer Applications CA, and Computer Science CS in a semester. Set CA 40 students opted for Java out of

60 students. Set CS 30 students opted for Java out of 50 students. Jaccard similarity coefficient Jjava (CA, CS) = 30/(60 + 50) x 100% = 27%. Two sets are sharing 27% of the members for Java course.

**Similarity of Document.**

An application ofJaccard similarity coefficient is in Natural Language Processing (NLP) and text processing. It quantifies the similarity in documents. Computational steps are as follows:

4. Find Bag of Words (Section 9.2.1.4) and remove words such as is, are, does, at, in, ....
5. Assign weighting factor is the Term frequency and Inverse Document Frequency (TF-IDF). Consider the frequency of words in the document.

6. Find k-shingles. A shingle is a word of fixed length. The k-shingles are the number of times the similar shingles extracted from a document or text. Examples of a shingle are Java, GP, 8.0, Python, 80%, Programming.
7. Find n-grams. A gram is a contiguous sequence of fixed length item (word
8. or set of characters, letters, words in pairs, triplets, quadruplets, ...) in a document or text. The n-grams are the number of times the similar items (1-grams, 2-grams, ..) extracted from a document or text. The 3-gram examples are lava GP 8.0, Python Programming 7.8, Big Data Analytics, 23A 240C 8LP, the numbers of which are extracted from the text.
9. Compute Jaccard similarity coefficient using Equation (6.22) between the documents.

**Collaborative Filtering as a Similar-Sets Finding Problem**

An analysis requires finding similar sets using collaborative filtering. Collaborative filtering refers to a filtering algorithm, which filters the items sets that have similarities with different items in a dataset.

CF finds the sets with items having the same or close similarity coefficients. Following are some examples of applications of CF:

Find those sets of students in computer application, and computer science who opt for the Java Programming subject in a semester.

Find sets of students in Java Programming subjects to whom same teacher taught and they showed excellent performance.

An algorithm finds the similarities between the sets for the CF. Applications of CF are in many ML methods, such as association rule mining, classifiers, and recommenders.

## Distance Measures for Finding Similar Items or Users

Distance can be defined in a number of ways. Distance is the measure of length of a line between two values in a two-dimensional map or graph. Set of Equations (6.20) measures distances.

For example, distance between (2014, 6%) and (2018, 8%) on a scatter plot when year is on the x axis and profit% on they axis is Distance = v [(2014 - 2018)2 + (6 - 8)2 ] = v (16 + 4) = 4.47, using Equation (6.20b). Distance can also be

similarly defined in v-dimensional space using Equation (6.20a).

Distances between all members in a set of points can be computed in metrics space using a mathematical equation. Metrics space means measurable or quantifiable space. For example, profit and year on a scatter plot are in metric space of two dimensions. Probability distribution function values are in metric space.

Consider student-performance measures 'very good' and 'excellent'. These parameters are in non-metric space. How are they made measurable? They become measurable when very good is specified as grade point average 8.5 which implies that a score between 8.0 to 9.0 is very good, and define 9.5 which implies that a score between 9.0 to 10.0 is excellent on a 10-point scale.

Euclidean distance $D_{Eu} = \left[\sum_{i=1}^{v} (x_i - x_i')^2\right]^{1/2}$

**Cosine distance**

Let **U** and **V** be two non-zero vectors, two documents in the vector space.

$$D_{Cos}(\mathbf{U}, \mathbf{V}) = \frac{\sum_i U_i V_i}{\sqrt{\sum_i U_i^2}\sqrt{\sum_i V_i^2}}$$

**Vector Cosine-Based Similarity Vector cosine similarity** in terms of angle between two vectors U and V is given by equation

$$\phi_{UV} = \cos^{-1}(\mathbf{U}, \mathbf{V}) = \frac{\boldsymbol{U} \cdot \boldsymbol{V}}{[\|\boldsymbol{U}\| - \|\boldsymbol{V}\|]}$$

**Concept of Sparse and Dense Vectors** Sparse vector uses a hash-map and consists of non-zero values. Hash-map is a collection, which stores data in (key- value) format (Section 3.3.1). Format is also called random access. Hashing means to convert a large value or string into shorter value or string so that indexing for searching is fast.

For example, assume a vector, which consists of array elements, (subject,

number of students opting, average GPA).

1. Dense vectors have elements (Hive, 40, 8.0), Oava, 30, 8.5), (FORTRAN, 0, 0), (Pascal, 0, 0). Dense vector consists of all elements, whether the element value is

O or not O.

2. Sparse vectors will be two only with elements (4, 40, 8.0) and (3, 30, 8.5). Random access Sparse vector means access to elements (key, value pairs) using key. Sparse vector consists of elements for which key is such that value is not O (Section 3.3.1).
3. Sparse vector has an associated hash-map in form of a hash-table. First row- Pascal, 1, second row- FORTRAN, 2, third row- Java, 3 and fourth row-Hive.
4. Hashing is a process of assigning a small number or small-sized string indexing, searching and memory saving purposes. Hash process uses a hash function, which results into not-colliding values. In case of two colliding numbers, the process assigns a new number. Sequential access sparse vectors mean two parallel accessing vectors, i.e., one to access keys and the other for values.

Edit distance DEd is a distance measure for dissimilarity between two set of strings or words. DEd equals the minimum number of inserts and deletes of characters needed to transform one set into another. Applications of edit distances are in text analytics and natural language processing, similarities in DNA sequences etc. DNA sequences are strings of characters.



**Hamming Distance**

If both U and V are vectors, Hamming distance DHa is equal to the number of

different elements between these two vectors. Recall Example 6.5 (iv) for Hamming distance between Jspi and Zspi. Hamming similarity-coefficient between car models Jaguar Land Rover and Zest is (1- 2/7) = 0.7. [70%]

Otta between two strings of equal length is the number of positions at which the corresponding characters differ. Otta is also equal to the minimum number of substitutions required to transform one string into the other. Otta is also equal to the minimum number of errors that need correction using transformation or substitution.

Hamming distance is therefore another distance measure for measuring the edit distance between two sets of strings, words or sequences.

**Frequent Item Set and Association rule Mining**

**Frequent Item Set**

Frequent itemset refers to a set of items that frequently appear together, for example, Python and Big Data Analytics. Students of computer science frequently choose these subjects for in-depth studies. Frequent itemset refers to a frequent itemset, which is a subset of items that appears frequently in a dataset.

Frequent Itemset Mining (FIM) refers to a data mining method which helps in discovering the itemsets that appear frequently in a dataset. For example, finding a set of students who frequently show poor performance in semester examinations. Frequent subsequence is a sequence of patterns that occurs frequently. For example, purchasing a football follows purchasing of sports kit. Frequent substructure refers to different structural forms, such as graphs, trees or lattices, which may be combined with itemsets or subsequences.

## Association Rule- Overview

An important method of data mining is association rule mining or association analysis. The method has been widely used in many application areas for discovering interesting relationships which are present in large datasets. The objective is to find uncovered relationships using some strong rules. The rules are termed as association rules for frequent itemsets. Mahout includes a 'parallel frequent pattern growth' algorithm. The method analyzes the items in a group and then identifies which items typically appear together (association)

## Apriori Algorithm

Apriori algorithm is used for frequent itemset mining and association rule mining. Apriori algorithm is considered as one of the most well-known association rule algorithms. The algorithm simply follows a basis that any subset of a large itemset must be a large itemset. This basis can be formally given as the Apriori principle. The Apriori principle can reduce the number of itemsets needed to be examined. Apriori principle suggests if an itemset is frequent, then all of its subsets must also be frequent. For example, if itemset {A, B, C} is a frequent itemset, then all of its subsets {A}, {B}, {C}, {A, B}, {B, C} and {A, C} must be frequent. On the contrary, if an itemset is not frequent, then none of its

supersets can be frequent. This results into a smaller list of potential frequent itemsets as the mining progresses.

Assume X and Y are two itemsets. Apriori principle holds due to the following property of support measure:

$$\forall\ X, Y: (X \subseteq Y) \rightarrow s(X) \geq s(Y)$$

Explanation: V means for all, and c means 'subset of' and can be 'equal to or included in'. Support of an itemset never exceeds the support of its subsets. This is known as the anti-monotone property of support.

The algorithm uses k-itemsets (An itemset which contains k items is known as a k-itemset) to explore (k+1)-itemsets in order to mine frequent itemsets from transactional database for the Boolean association rules (If Then rule is a Boolean association rule, as it checks if true or false).

The frequent itemset algorithm uses candidate generation process. The groups of candidates are then tested against the dataset. Apriori uses breadth-first search method and a hash tree structure to count candidate itemsets. Also, it is assumed that items within an itemset are kept in lexicographic order. The algorithm identifies the frequent individual items in the database and extends them to larger and larger itemsets as long as those itemsets are found in the database. The frequent itemsets provide the general trends in the database as well.

**Evaluation of Candidate rules**

Apriori algorithm evaluates candidates for association as follows:

$C_k$: Set of candidate-itemsets of size $k$

$F_l$: Set of frequent itemsets of size $k$

$F_1$ = {large items}

*for* (k=1; $F_k$ != 0; k++) *do* {

$C_{k+1}$ = New candidates generated from $F_k$

*for* each transaction t in the database *do*

Increment the count of all candidates in Ck+1 that are contained int Fk+i = Candidates in Ck+i with minimum support

}

Steps of the algorithm can be stated in the following manner:

Candidate itemsets are generated using only large itemsets of the previous iteration. The transactions in the database are not considered while generating candidate itemsets.

The large itemset of the previous iteration is joined with itself to generate all itemsets having size higher by 1.

Each generated itemset that does not have a large subset is discarded. The remaining itemsets are candidate itemsets.

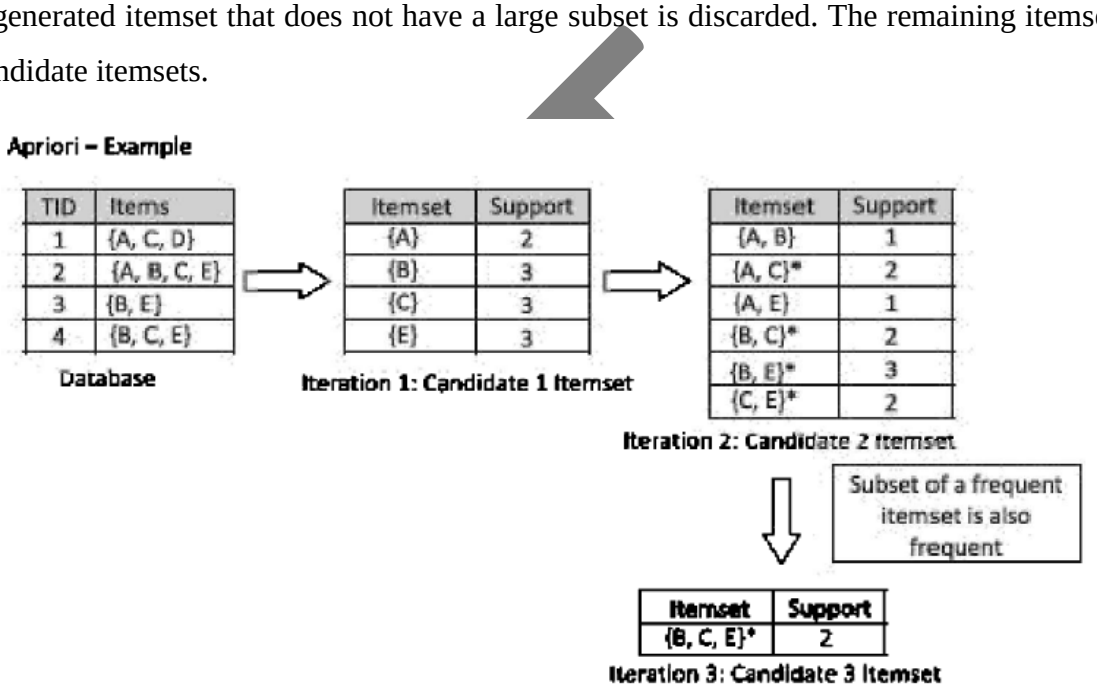



Figure shows Apriori algorithm process for adopting the subset of frequent itemsets as a frequent itemset.

It is observed in the Apriori example that every subset of a frequent itemset is also frequent. Thus, a candidate itemset in Ck+1 can be pruned even if one of its subsets is not contained in Fk.

**Applications of Association rule mining**

**Market basket analysis** is a tool for knowledge discovery about co-occurrence of items. A co-occurrence means two or more things occur together. It can also be defined as a data mining technique to derive the strength of association between pairs of product items. If

people tend to buy two products (say A and B) together, then the buyer of product A is a potential customer for an advertisement of product B.

The concept is similar to the real market basket where we select an item (product) and put it in a basket (itemset). The basket symbolizes the transactions. The number of baskets is very high as compared to the items in a basket. A set of items that is present in many baskets is termed as a frequent itemset. Frequency is the proportion of baskets that contain the items of interest.

Market basket analysis can be applied to many areas. The following example explains the market basket model using application examples.

Market basket analysis generates If-Then scenario rules. For example, if X occurs then Y is likely to occur too. If item A is purchased, then item B is likely to be purchased too. The rules are derived from the experience. This may be the result of frequencies of co-occurrence of items in past transactions.

The rules can be used in several analytical strategies. The rules can be written in format If {A} Then {B}. The If part of the rule (A) is known as antecedent and the THEN part of the rule (B) is known as consequent. The condition is antecedent and the result is consequent.

The applications of market basket analysis in various domains other than retail are:

- Medical analytics: Market basket analysis can be used for conditions and symptom analysis. This helps in identifying a profile of illness in a better way. The analysis is also useful in genome analysis, molecular fragment mining, drug design and studying the role of biomarkers in medicine. The analysis can also help to reveal biologically relevant associations betweendifferent genes. Further, it can also help to find the effect of environment on gene expressions.
- Web usage analytics: FIM approaches can be used with viewing data on websites. The information contained in association rules can be exploited to learn about website browsing of visitor's behavior, developing website structure by making it more effective for visitors, or improving web marketing promotions. The results of this type of analysis can be used to inform website design (how items are grouped together) and to power recommendation engines (Section 6.8). Results are helpful in targeted marketing. For example, advertising content that people are probably interested in, based on past behavior of users.
- Fraud detection and technical dependence analysis: Extract knowledge so that

normal behavior patterns may be obtained in illegal transactions from a credit card database in order to detect and prevent fraud. Another example can be to find frequently occurring relationships or FIM rules

between the various parties involved in the handling of the financial claim. Some examples are:

- Financial institutions to analyze credit card purchases of customers to build profiles for fraud detection purposes and cross-selling opportunities.
- Insurance institution builds the profiles to detect insurance claim fraud. The profiles of claims help to determine if more than one claim belongs to a particular victim within a specified period of time.

- Click stream analysis or web link analysis: Click stream refers to a sequence of web pages viewed by a user. Analysis of clicks is the process of extracting knowledge from web logs. This helps to discover the unknown and potentially interesting patterns useful in the future. It facilitates an understanding of the behavior of website visitors. This knowledge can be used to enhance the way that web pages are interconnected or for increasing the sales of the commercial websites.
- Telecommunication services analysis: Market basket analysis can be used to determine the type of services being utilized and the packages customers are purchasing. This knowledge can be used to plan marketing strategies for customers who are interested in similar services. For example, telecommunication companies can offer TV Internet, and web- services by creating combined offers. The analysis might also be useful to determine capacity requirements.
- Plagiarism detection: It is the process of locating instances of similar content or idea within a work or a document. Plagiarism detection can findsimilarities among statements that may lead to similar paragraphs if all statements are similar and that possibly lead to similar documents. Formation of relevant word and sentence sequences for detection of plagiarism using association rule mining technique is also very popular technique.

## Finding Associations

- Association rules intend to tell how items of a dataset are associated with each other. The concept of association rules was introduced in 1993 for discovering relations between

items in sales data of a large retailing company. Association analysis is applicable to several domains. Some of them are marketing, bioinformatics, web mining, scientific data analysis, and intrusion detection systems.

- The applications might be to find: products that are often purchased together, types of DNA sensitive to a new drug, the possibility of classifying web documents automatically, geophysical trends or patterns in seismicity to predict earthquakes and automate the malicious detecting characteristics.
- In medical diagnosis, for example, considering the co-morbid (co-occur) conditions can help in treating the patient in better way. This helps in improving patient care and medicine prescription.

**Finding Similarity**

Let A and B be two itemsets. Jaccard similarity index of two itemsets is measured in terms of set theory using the following equation:

$$\text{Jaccard itemsets similarity index} = 1 - \frac{|A \cap B|}{|A \cup B|} \times 100\%.$$

Explanation: n means intersection, number of those elements or items which are the same in set A and B. U means union, number of elements or items present in union of A and B.

### How will you define a similarity in a purchase of car model

Assume two sets of car customers, youth Y and family F. Assume in set Y, 40 out of 100 youths and F 50 out of 200 families opted for the Tata Zest car model. Jaccard similarity index Jzest (Y, F) = 40/ (100 + 200).100% = 13%. Two sets are sharing 13% of the members who purchased a Zest.

# MODULE 5

# Chapter 1: Text Mining

Text mining is the art and science of discovering knowledge, insights and patterns from an organized collection of textual databases. Textual mining can help with frequency analysis of important terms, and their semantic relationships.

Text is an important part of the growing data in the world. Social media technologies have enabled users to become producers of text and images and other kinds of information. Text mining can be applied to large-scale social media data for gathering preferences, and measuring emotional sentiments. It can also be applied to societal, organizational and individual scales.

## 1.1 Text Mining Applications

Text mining is a useful tool in the hands of chief knowledge officers to extract knowledge relevant to an organization. Text mining can be used across industry sectors and application areas, including decision support, sentiment analysis, fraud detection, survey analysis, and many more.

1. *Marketing*: The voice of the customer can be captured in its native and raw format and then analyzed for customer preferences and complaints.
    1. Social personas are a clustering technique to develop customer segments of interest. Consumer input from social media sources, such as reviews, blogs, and tweets, contain numerous leading indicators that can be used towards anticipating and predicting consumer behavior.
    2. A 'listening platform' is a text mining application, that in real time, gathers social media, blogs, and other textual feedback, and filters out the chatter to extract true consumer sentiment. The insights can lead to more effective product marketing and better customer service.

2. The customer call center conversations and records can be analyzed for patterns of customer complaints. Decision trees can organize this data to create decision choices that could help with product management activities and to become proactive in avoiding those complaints.

3. *Business operations*: Many aspects of business functioning can be accurately gauged from analyzing text./
    1. Social network analysis and text mining can be applied to emails, blogs, social media and other data to measure the emotional states and the mood of employee populations. Sentiment analysis can reveal early signs of employee dissatisfaction which can then can be proactively managed.

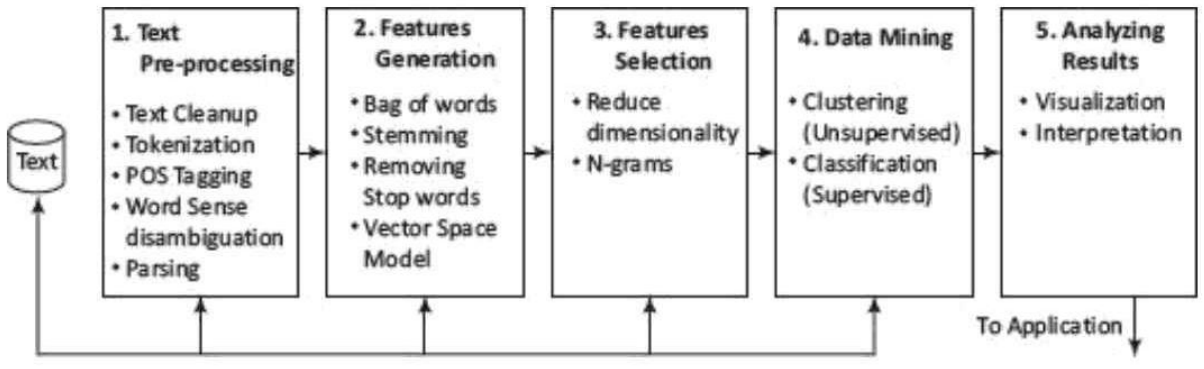
2. Studying people as emotional investors and using text analysis of the social Internet to measure mass psychology can help in obtaining superior investment returns.

3. *Legal*: In legal applications, lawyers and paralegals can more easily search case histories and laws for relevant documents in a particular case to improve their chances of winning.
    1. Text mining is also embedded in e-discovery platforms that help in minimizing risk in the process of sharing legally mandated documents.
    2. Case histories, testimonies, and client meeting notes can reveal additional information, such as morbidities in a healthcare situation that can help better predict high-cost injuries and preventcosts.

4. Governance and Politics: Governments can be overturned based on a tweet originating from a self-immolating fruit-vendor inTunisia.
    1. Social network analysis and text mining of large-scale social media data can be used for measuring the emotional states and the mood of constituent populations. Micro-targeting constituents with specific messages gleaned from social media analysis can be a more efficient use of resources when fighting democratic elections.
    2. In geopolitical security, internet chatter can be processed for real- time information and to connect the dots on any emerging threats.
    3. In academic, research streams could be meta-analyzed for underlying research trends.

## 1.2 Text Mining Process

Text Mining is a rapidly evolving area of research. As the amount of social media and other text data grows, there is need for efficient abstraction and categorization of meaningful information from the text.




The five phases for processing text are as follows:

**Phase 1: Text pre-processing** enables Syntactic/Semantic text-analysis and does the followings:

1. Text *cleanup* is a process of removing unnecessary or unwanted information. Text cleanup converts the raw data by filling up the missing values, identifies and removes outliers, and resolves the inconsistencies. For example, removing comments, removing or escaping "%20" from URL for the web pages or cleanup the typing error, such as teh (the), do n't (do not) [%20 specifies space in a URL].
2. *Tokenization* is a process of splitting the cleanup text into tokens (words) using white spaces and punctuation marks as delimiters.
3. *Part of Speech (POS) tagging* is a method that attempts labeling of each token (word) with an appropriate POS. Tagging helps in recognizing names of people, places, organizations and titles. English language set includes the noun, verb, adverb, adjective,

   prepositions and conjunctions. Part of Speech encoded in the annotation system of the Penn Treebank Project has 36 POS tags.4
4. *Word sense disambiguation* is a method, which identifies the sense of a word used in a sentence; that gives meaning in case the word has multiple meanings. The methods, which resolve the ambiguity of words can be context or proximity based. Some examples of such words are bear, bank, cell and bass.
5. *Parsing* is a method, which generates a parse-tree for each sentence. Parsing attempts and infers the precise grammatical relationships between different words in a given sentence.

**Phase 2: Features Generation** is a process which first defines features (variables, predictors). Some of the ways of feature generations are:

1. *Bag of* words-Order of words is not that important for certain applications.

   Text document is represented by the words it contains (and their occurrences). Document classification methods commonly use the bag-of-words model. The pre-processing of a document first provides a document with a bag of words. Document classification methods then use the occurrence (frequency) of each word as a feature for training a classifier. Algorithms do not directly apply on the bag of words, but use the frequencies.
2. Stemming-identifies a word by its root.

   (i) Normalizes or unifies variations of the same concept, such as *speak* for three variations, i.e., speaking, speaks, speakers denoted by [speaking, speaks, speaker- + speak]

   (ii) Removes plurals, normalizes verb tenses and remove affixes.

   Stemming reduces the word to its most basic element. For example, impurification -+ pure.
3. *Removing stop words* from the feature space-they are the common words, unlikely to help text mining. The search program tries to ignore stop words. For example, ignores *a, at, for,* it, *in*
   and *are.*
4. *Vector Space Model* (VSM)-is an algebraic model for representing text documents as vector of identifiers, word frequencies or terms in the document index. VSM uses the method of term frequency-inverse document frequency (TF-IDF) and evaluates how important is a

word in a document.

When used in document classification, VSM also refers to the bag-of-words model. This bag of words is required to be converted into a term-vector in VSM. The term vector provides the numeric values corresponding to each term appearing in a document. The term vector is very helpful in feature generation and selection.

*Term frequency and inverse document frequency (IDF)* are important metrics in text analysis. TF-IDF weighting is most common- Instead of the simple TF, IDF is used to weight the importance of word in the document.

**Phase 3: Features Selection** is the process that selects a subset of features by rejecting irrelevant and/or redundant features (variables, predictors or dimension) according to defined criteria. Feature selection process does the following:

1. *Dimensionality* reduction-Feature selection is one of the methods of division and therefore, dimension reduction. The basic objective is to eliminate irrelevant and redundant data. Redundant features are those, which provide no extra information. Irrelevant features provide no useful or relevant information in any context.

   Principal Component Analysis (PCA) and Linear Discriminate Analysis (LDA) are dimension reduction methods. Discrimination ability of a feature measures relevancy of features. Correlation helps in finding the redundancy of the feature. Two features are redundant to each other if their values correlate with each other.

2. *N-gram* evaluation-finding the number of consecutive words of interest and extract them. For example, 2-gram is a two words sequence, ["tasty food", "Good one"]. 3-gram is a three words sequence, ["Crime Investigation Department"].

3. *Noise detection and evaluation of outliers* methods do the identification of unusual or suspicious items, events or observations from the data set. This step helps in cleaning the data.

The feature selection algorithm reduces dimensionality that not only improves the performance of learning algorithm but also reduces the storage requirement for a dataset. The process enhances data understanding and its visualization.

**Phase 4: Data mining techniques** enable insights about the structured database that resulted from the previous phases. Examples of techniques are:

1. Unsupervised learning (for example, clustering)

   (i) The class labels (categories) of training data are unknown

   (ii) Establish the existence of groups or clusters in the data

   Good clustering methods use high intra-cluster similarity and low inter-cluster similarity. Examples of uses - biogs, pattern

   and trends.

2. *Supervised learning (for example, classification)*

   (i) The training data is labeled indicating the class

   (ii) New data is classified based on the training set

   Classification is correct when the known label of test sample is identical with the resulting class computed from the classification model.

Examples of uses are *news filtering application,* where it is required to automatically assign incoming documents to pre-defined categories; *email spam filtering,* where it is identified whether incoming email messages are spam or not.

Example of text classification methods are *Naive Bayes Classifier* and *SVMs.*

3. *Identifying evolutionary patterns* in temporal text streams-the method is useful in a wide range of applications, such as summarizing of events in news articles and extracting the research trends in the scientific literature.

**Phase 5: Analysing results**

(i) Evaluate the outcome of the complete process.

(ii) Interpretation of Result- If acceptable then results obtained can be used as an input for next set of sequences. Else, the result can be discarded, and try to understand what and why the process failed.

(iii) Visualization - Prepare visuals from data, and build a prototype.

(iv) Use the results for further improvement in activities at the enterprise, industry or institution.

## Text Mining Challenges

The challenges in the area of text mining can be classified on the basis of documents area-characteristics. Some of the classifications are as follows:

1. NLP issues:
   (i) POS Tagging
   (ii) Ambiguity
   (iii) Tokenization
   (iv) Parsing
   (v) Stemming
   (vi) Synonymy and polysemy
2. Mining techniques:
   (i) Identification of the suitable algorithm(s)
   (ii) Massive amount of data and annotated corpora
   (iii) Concepts and semantic relations extraction
   (iv) When no training data is available
3. Variety of data:
   (i) Different data sources require different approaches and different areas of expertise

(ii) Unstructured and language independency

4. Information visualization

5. Efficiency when processing real-time text stream

6. Scalability

## 1.3 Term Document Matrix

This is the heart of the structuring process. Free flowing text can be transformed into numeric data in a TDM, which can then be mined using regular data mining techniques.

1. There are several efficient techniques for identifying key terms from a text. There are less efficient techniques available for creating topics out of them. For the purpose of this discussion, one could call key words, phrases or topics as a term of interest. This approach measures the frequencies of select important terms occurring in each document. This creates a t x d Term–by–Document Matrix (TDM) where t is the number of terms and d is the number of documents (Table 11.1).
2. Creating a TDM requires making choices of which terms to include. The terms chosen should reflect the stated purpose of the text mining exercise. The list of terms should be as extensive as needed, but should not include unnecessary stuff that will serve to confuse the analysis, or slow the computation.



| | Term-Document Matrix | | | | |
|---|---|---|---|---|---|
| Document/ Terms | Investment | Profit | Happy | Success | … |
| Doc 1 | 10 | 4 | 3 | 4 | |
| Doc 2 | 7 | 2 | 2 | | |
| Doc 3 | | | 2 | 6 | |
| Doc 4 | 1 | 5 | 3 | | |
| Doc 5 | | 6 | | 2 | |
| Doc 6 | 4 | | 2 | | |
| … | | | | | |

**Table 1.1: Term-Document Matrix**

Here are some considerations in creating a TDM.

1. A large collection of documents mapped to a large bag of words will likely lead to a very sparse matrix if they have few common words. Reducing dimensionality of data will help improve the speed of analysis and meaningfulness of the results. Synonyms, or terms will similar meaning, should be combined and should be counted together, as a common term. This would help reduce the number of distinct terms of words or 'tokens'.
2. Data should be cleaned for spelling errors. Common spelling errors should be ignored and the terms should be combined. Uppercase- lowercase terms should also be combined.
3. When many variants of the same term are used, just the stem of theword would be used to reduce the number of terms. For instance, terms like customer order, ordering, order data, should be combined into a single token word, called 'Order'.
4. On the other side, homonyms (terms with the same spelling but different meanings) should be counted separately. This would enhance the quality of analysis. For example, the term order can mean a customer order, or the ranking of certain choices. These two should be treated separately. "The boss ordered that the customer orders data analysis be presented in chronological order'. This statement shows three different meanings for the word 'order'. Thus, there will be a need for a manual review of the TD matrix.
5. Terms with very few occurrences in very few documents should be eliminated from the matrix. This would help increase the density of the matrix and the quality of analysis.
6. The measures in each cell of the matrix could be one of several possibilities. It could be a simple count of the number of occurrences of each term in a document. It could also be the log of that number. It could be the fraction number computed by dividing the frequency count by the total number of words in the document. Or there may be binary values in the matrix to represent whether a term is mentioned or not. The choice of value in the cells will depend upon the purpose of the textanalysis.



At the end of this analysis and cleansing, a well-formed, densely populated, rectangular, TDM will be ready for analysis. The TDM could be mined using all the available data mining techniques.

## 1.4 Mining the TDM

The TDM can be mined to extract patterns/knowledge. A variety of techniques could be applied to the TDM to extract new knowledge.

Predictors of desirable terms could be discovered through predictive techniques, such as regression analysis. Suppose the word profit is a desirable word in a document. The number of occurrences of the word profit in a document could be regressed against many other terms in the TDM. The relative strengths of the coefficients of various predictor variables would

show the relative impact of those terms on creating a profitdiscussion.

Predicting the chances of a document being liked is another form of analysis. For example, important speeches made by the CEO or the CFO to investors could be evaluated for quality. If the classification of those documents (such as good or poor speeches) was available, then the terms of TDM could be used to predict the speech class. A decision tree could be constructed that makes a simple tree with a few decision points that predicts the success of a speech 80 percent of the time. This tree could be trained with more data to become better over time.

Clustering techniques can help categorize documents by common profile. For example, documents containing the words investment and profit more often could be bundled together. Similarly, documents containing the words, customer orders and marketing, more often could be bundled together. Thus, a few strongly demarcated bundles could capture the essence of the entire TDM. These bundles could thus help with further processing, such as handing over select documents to others for legal discovery.

Association rule analysis could show relationships of coexistence. Thus, one could say that the words, tasty and sweet, occur together often (say 5 percent of the time); and further, when these two words are present, 70 percent of the time, the word happy, is also present in the document.



## 1.5 Comparing Text Mining and Data Mining

Text Mining is a form of data mining. There are many common elements between Text and Data Mining. However, there are some key differences (Table 1.2). The key difference is that text mining requires conversion of text data into frequency data, before data mining techniques can beapplied.

| Dimension | **Text Mining** | **Data Mining** |
|---|---|---|
| Nature of data | Unstructured data: Words, phrases, sentences | Numbers; alphabetical and logical values |
| Language used | Many languages and dialects used in the world; many languages are extinct, new documents are discovered | Similar numerical systems across the world |
| Clarity and precision | Sentences can be ambiguous; sentiment may contradict the words | Numbers are precise. |
| Consistency | Different parts of the text can contradict each other | Different parts of data can be inconsistent, thus, requiring statistical significance analysis |
| Sentiment | Text may present a clear and consistent or mixed sentiment, across a continuum. Spoken words adds further sentiment | Not applicable |
| Quality | Spelling errors. Differing values of proper nouns such as names. Varying quality of language translation | Issues with missing values, outliers, etc |
| Nature of Analysis | Keyword based search; co- existence of themes; Sentiment Mining | A full wide range of statistical and machine learning analysis for relationship and differences |

**Table 1.2: Comparing Text Mining and Data Mining**

## 1.6 Text Mining Best Practices

Many of the best practices that apply to the use of data mining techniques will also apply to text mining.

1. The first and most important practice is to ask the right question. A good question is one which gives an answer and would lead to large payoffs for the organization. The purpose and the key question will define how and at what levels of granularity the TDM would be made. For example, TDM defined for simpler searches would be different from those used for complex semantic analysis or network analysis.
2. A second important practice is to be creative and open in proposing imaginative hypotheses for the solution. Thinking outside the box is important, both in the quality of the proposed solution as well as in finding the high quality data sets required to test the hypothesized solution. For example, a TDM of consumer sentiment data should be combined with customer order data in order to develop a comprehensive view of customer behavior. It's important to assemble a team that has a healthy mix of technical and business skills.
3. Another important element is to pursue the problem iteratively. Too much data can overwhelm the infrastructure and also befuddle the mind. It is better to divide and conquer the problem with a simpler TDM, with fewer terms and fewer documents and data sources. Expand as needed, in an iterative sequence of steps. In the future, add new terms to help improve predictive accuracy.
4. A variety of data mining tools should be used to test the relationships in the TDM. Different decision tree algorithms could be run alongside cluster analysis and other techniques. Triangulating the findings with multiple techniques, and many what-if scenarios, helps build confidence in the solution. Test the solution in many ways before committing to deploy it.

# Chapter 2: Web Mining

Web mining is the art and science of discovering patterns and insights from the World-wide web so as to improve it. The world-wide web is at the heart of the digital revolution. More data is posted on the web every day than was there on the whole web just 20 years ago. Billions of users are using it every day for a variety of purposes. The web is used for electronic commerce, business communication, and many other applications. Web mining analyzes data from the web and helps find insights that could optimize the web content and improve the user experience. Data for web mining is collected via Web crawlers, web logs, and other means.

Here are some characteristics of optimized websites:

1. *Appearance*: Aesthetic design. Well-formatted content, easy to scan and navigate. Good color contrasts.
2. *Content*: Well planned information architecture with useful content. Fresh content. Search-engine optimized. Links to other goodsites.
3. *Functionality*: Accessible to all authorized users. Fast loading times. Usable forms. Mobile enabled.

This type of content and its structure is of interest to ensure the web is easy to use. The analysis of web usage provides feedback on the web content, and also the consumer's browsing habits. This data can be of immense use for commercial advertising, and even for social engineering.

The web could be analyzed for its structure as well as content. The usage pattern of web pages could also be analyzed. Depending upon objectives, web mining can be divided into three different types: Web usage mining, Web content mining and Web structure mining (Figure 2.1).

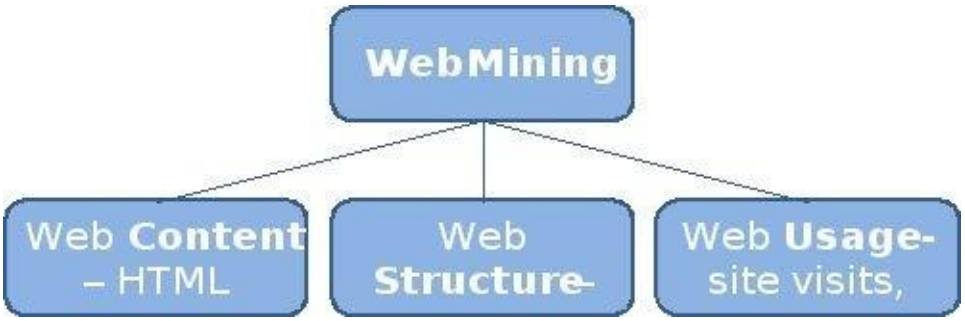



**Figure: 2.1 Web Mining structure**

## 2.1 Web content mining

A website is designed in the form of pages with a distinct URL (universal resource locator). A large website may contain thousands of pages. These pages and their content is managed using specialized software systems called Content Management Systems. Every page can have text, graphics, audio, video, forms, applications, and more kinds of content including user generated content.

The websites keep a record of all requests received for its page/URLs, including the requester information using 'cookies'. The log of these requests could be analyzed to gauge the popularity of those pages among different segments of the population. The text and application content on the pages could be analyzed for its usage by visit counts. The pages on a website themselves could be analyzed for quality of content that attracts most users. Thus the unwanted or unpopular pages could be weeded out, or they can be transformed with different content and style. Similarly, more resources could be assigned to keep the more popular pages more fresh and inviting.

## 2.2 Web structure mining

The Web works through a system of hyperlinks using the hypertext protocol (http). Any page can create a hyperlink to any other page, it can be linked to by another page. The intertwined or self-referral nature of web lends itself to some unique network analytical algorithms. The structure of Web pages could also be analyzed to examine the pattern of hyperlinks among pages. There are two basic strategic models for successful websites: Hubs and Authorities.

1. *Hubs*: These are pages with a large number of interesting links. They serve as a hub, or a gathering point, where people visit to access a variety of information. Media sites like Yahoo.com, or government sites would serve that purpose. More focused sites like Traveladvisor.com and yelp.com could aspire to becoming hubs for new emerging areas.
2. *Authorities*: Ultimately, people would gravitate towards pages that provide the most complete and authoritative information on a particular subject. This could be factual information, news, advice, user reviews etc. These websites would have the most number of inbound links from other websites. Thus Mayoclinic.com would serve as an authoritative page for expert medical opinion. NYtimes.com would serve as an authoritative page for daily news.

## 2.3 Web usage mining

As a user clicks anywhere on a webpage or application, the action is recorded by many entities in many locations. The browser at the client machine will record the click, and the web server providing the content would also make a record of the pages served and the user activity on those pages. The entities between the client and the server, such as the router, proxy server, or ad server, too would record that click.

The goal of web usage mining is to extract useful information and patterns from data generated through Web page visits and transactions. The activity data comes from data stored

in server access logs, referrer logs, agent logs, and client-side cookies. The user characteristics and usage profiles are also gathered directly, or indirectly, through syndicated data. Further, metadata, such as page attributes, content attributes, and usage data are also gathered.

The web content could be analyzed at multiple levels (Figure 2.2).

1. The *server side analysis* would show the relative popularity of the web pages accessed. Those websites could be hubs and authorities.
2. The *client side analysis* could focus on the usage pattern or the actual content consumed and created by users.
    1. Usage pattern could be analyzed using 'clickstream' analysis, i.e. analyzing web activity for patterns of sequence of clicks, and the location and duration of visits on websites. Clickstream analysis can be useful for web activity analysis, software testing, market research, and analyzing employee productivity.
    2. Textual information accessed on the pages retrieved by users could be analyzed using text mining techniques. The text would be gathered and structured using the bag-of-words technique to build a Term-document matrix. This matrix could then be mined using cluster analysis and association rules for patterns such as popular topics, user segmentation, and sentiment analysis.

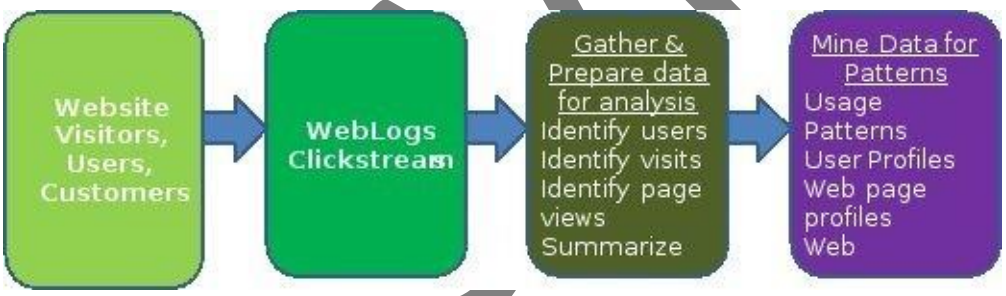



**Figure: 2.2 Web Usage Mining architecture**

Web usage mining has many business applications. It can help predict user behavior based on previously learned rules and users' profiles, and can help determine lifetime value of clients. It can also help design cross-marketing strategies across products, by observing association rules among the pages on the website. Web usage can help evaluate promotional campaigns and see if the users were attracted to the website and used the pages relevant to the campaign. Web usage mining could be used to present dynamic information to users based on their interests and profiles. This includes targeted online ads and coupons at user groups based on user access patterns.

## 2.4 Web Mining Algorithms

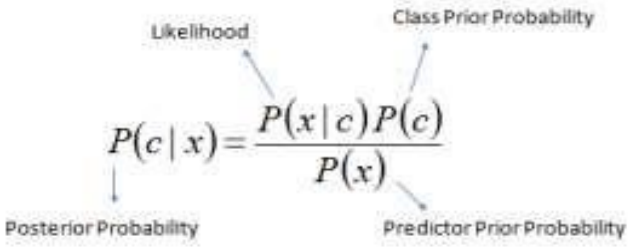
Hyperlink-Induced Topic Search (HITS) is a link analysis algorithm that rates web pages as being hubs or authorities. Many other HITS-based algorithms have also been published. The most famous and powerful of these algorithms is the PageRank algorithm. Invented by

Google co-founder Larry Page, this algorithm is used by Google to organize the results of its search function. This algorithm helps determine the relative importance of any particular web page by counting the number and quality of links to a page. The websites with more number of links, and/or more links from higher-quality websites, will be ranked higher. It works in a similar way as determining the status of a person in a society of people. Those with relations to more people and/or relations to people of higher status will be accorded a higher status.

PageRank is the algorithm that helps determine the order of pages listed upon a Google Search query. The original PageRank algorithm formuation has been updated in many ways and the latest algorithm is kept a secret so other websites cannot take advantage of the algorithm and manipulate their website according to it. However, there are many standard elements that remain unchanged. These elements lead to the principles for a good website. This process is also called Search Engine Optimization (SEO).

# Chapter 3
# Naïve Bayes Analysis

Naïve Bayes technique is a is supervised machine learning technique that that uses probability theory based analysis.
It is machine learning technique that computes the probabilities of an instance of belonging to each of many target classes, given the prior probabilities of classification using individual factors.

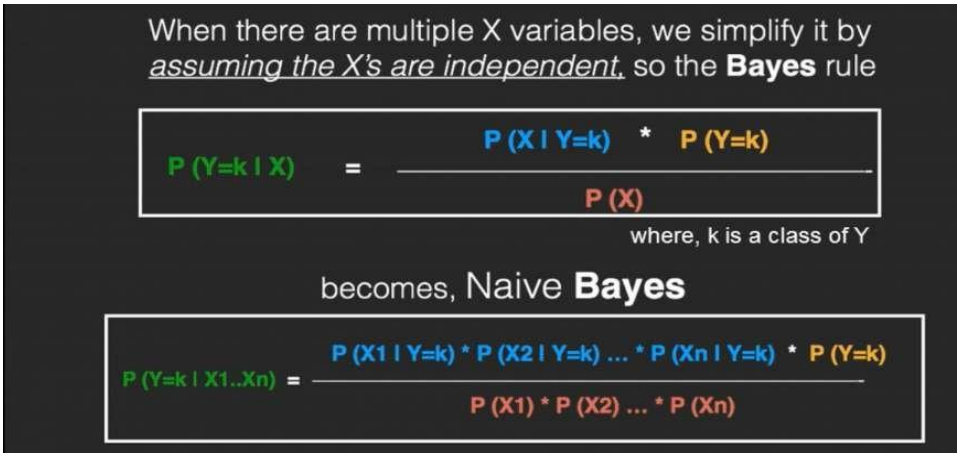


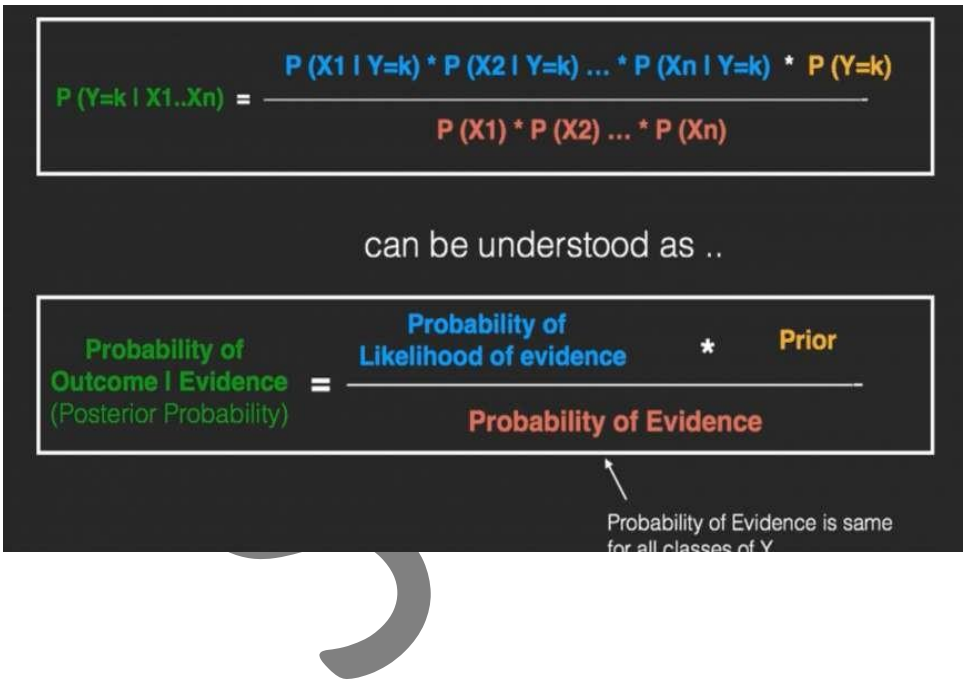

$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \cdots \times P(x_n|c) \times P(c)$$

- P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).
- P(c) is the prior probability of class.
- P(x|c) is the likelihood which is the probability of predictor given class.
- P(x) is the prior probability of predictor.

## 3.1 Probability

- The Bayes Rule provides the formula for the probability of Y given X. But, in real-world problems, you typically have multiple X variables.
- When the features are independent, we can extend the Bayes Rule to what is called Naive Bayes.
- It is called 'Naive' because of the naive assumption that the X's are independent of each other. Regardless of its name, it's a powerful formula.




- In technical jargon, the left-hand-side (LHS) of the equation is understood as the posterior probability or simply the posterior.
- The RHS has 2 terms in the numerator.
- The first term is called the 'Likelihood of Evidence'. It is nothing but the conditional probability of each X's given Y is of particular class 'c'.
- Since all the X's are assumed to be independent of each other, you can just multiply the 'likelihoods' of all the X's and called it the 'Probability of likelihood of evidence'. This is known from the training dataset by filtering records where Y=c.
- The second term is called the prior which is the overall probability of Y=c, where c is a class of Y. In simpler terms, Prior = count(Y=c) / n_Records.

- An example is better than an hour of theory. So let's see one

## Naive Bayes Example

- Say you have 1000 fruits which could be either 'banana', 'orange' or 'other'.
- These are the 3 possible classes of the Y variable.
- We have data for the following X variables, all of which are binary (1 or 0).
- Long
- Sweet
- Yellow
- The first few rows of the training dataset look like this:

| Fruit | Long (x1) | Sweet (x2) | Yellow (x3) |
|---|---|---|---|
| Orange | 0 | 1 | 0 |
| Banana | 1 | 0 | 1 |
| Banana | 1 | 1 | 1 |
| Other | 1 | 1 | 0 |

- For the sake of computing the probabilities, let's aggregate the training data to form a counts table like this.

| Type | Long | Not Long | Sweet | Not Sweet | Yellow | Not Yellow | Total |
|---|---|---|---|---|---|---|---|
| Banana | 400 | 100 | 350 | 150 | 450 | 50 | 500 |
| Orange | 0 | 300 | 150 | 150 | 300 | 0 | 300 |
| Other | 100 | 100 | 150 | 50 | 50 | 150 | 200 |
| Total | 500 | 500 | 650 | 350 | 800 | 200 | 1000 |

- So the objective of the classifier is to predict if a given fruit is a 'Banana' or 'Orange' or 'Other' when only the 3 features (long, sweet and yellow) are known.
- Let's say you are given a fruit that is: Long, Sweet and Yellow, can you predict what fruit it is?
- This is the same of predicting the Y when only the X variables in testing data are known. Let's solve it by hand using Naive Bayes.
- The idea is to compute the 3 probabilities, that is the probability of the fruit being a banana, orange or other. Whichever fruit type gets the highest probability wins.
- All the information to calculate these probabilities is present in the above tabulation.
- Step 1: Compute the 'Prior' probabilities for each of the class of fruits.
    - P(Y=Banana) = 500 / 1000 = 0.50
    - P(Y=Orange) = 300 / 1000 = 0.30
    - P(Y=Other) = 200 / 1000 = 0.20
- Step 2: Compute the probability of evidence that goes in the denominator.
    - P(x1=Long) = 500 / 1000 = 0.50
    - P(x2=Sweet) = 650 / 1000 = 0.65
    - P(x3=Yellow) = 800 / 1000 = 0.80
- Step 3: Compute the probability of likelihood of evidences that goes in the numerator.
    - Here, I have done it for Banana alone.
    - Probability of Likelihood for Banana
    - P(x1=Long | Y=Banana) = 400 / 500 = 0.80
    - P(x2=Sweet | Y=Banana) = 350 / 500 = 0.70

- P(x3=Yellow | Y=Banana) = 450 / 500 = 0.90

- Step 4: Substitute all the 3 equations into the Naive Bayes formula, to get the probability that it is a banana.


## Advantages

- When assumption of independent predictors holds true, a Naive Bayes classifier performs better as compared to other models
- Naive Bayes requires a small amount of training data to estimate the test data. So, the training period is less.
- Naive Bayes is also easy to implement.

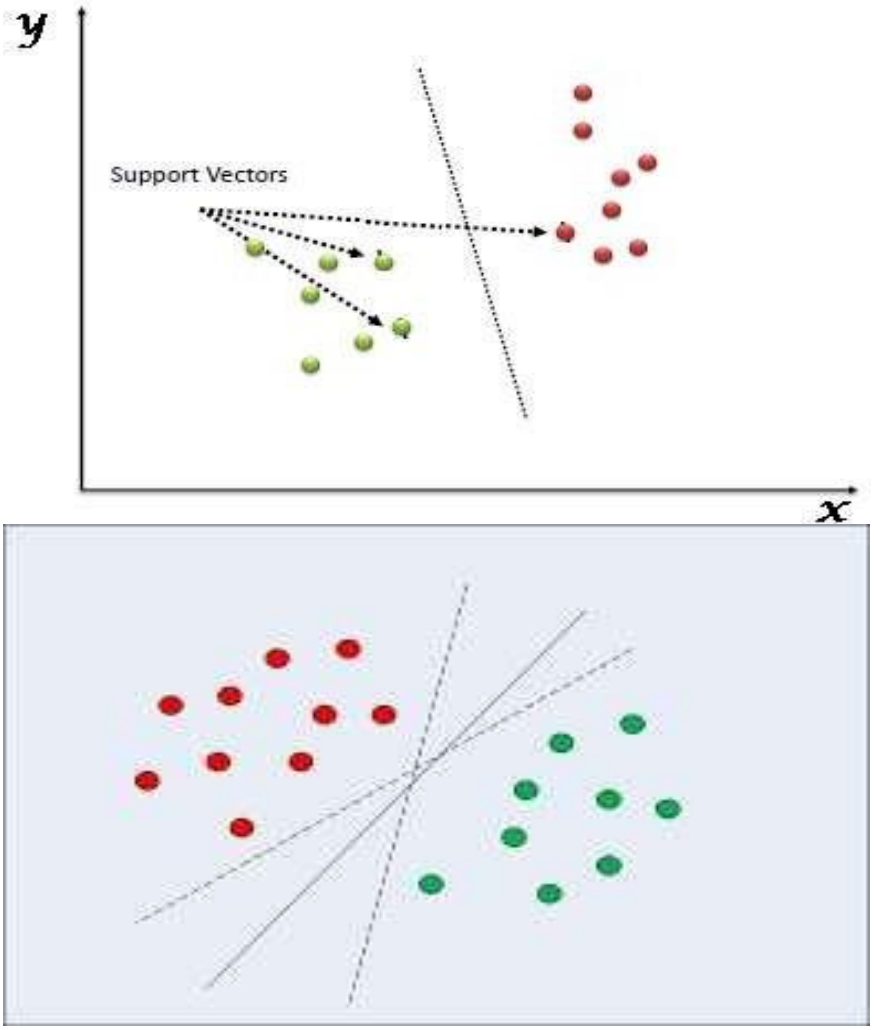
## Disadvantages

- Main imitation of Naive Bayes is the assumption of independent predictors. Naive Bayes implicitly assumes that all the attributes are mutually independent. In real life, it is almost impossible that we get a set of predictors which are completely independent.
- If categorical variable has a category in test data set, which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as Zero Frequency. To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.

# Chapter 5

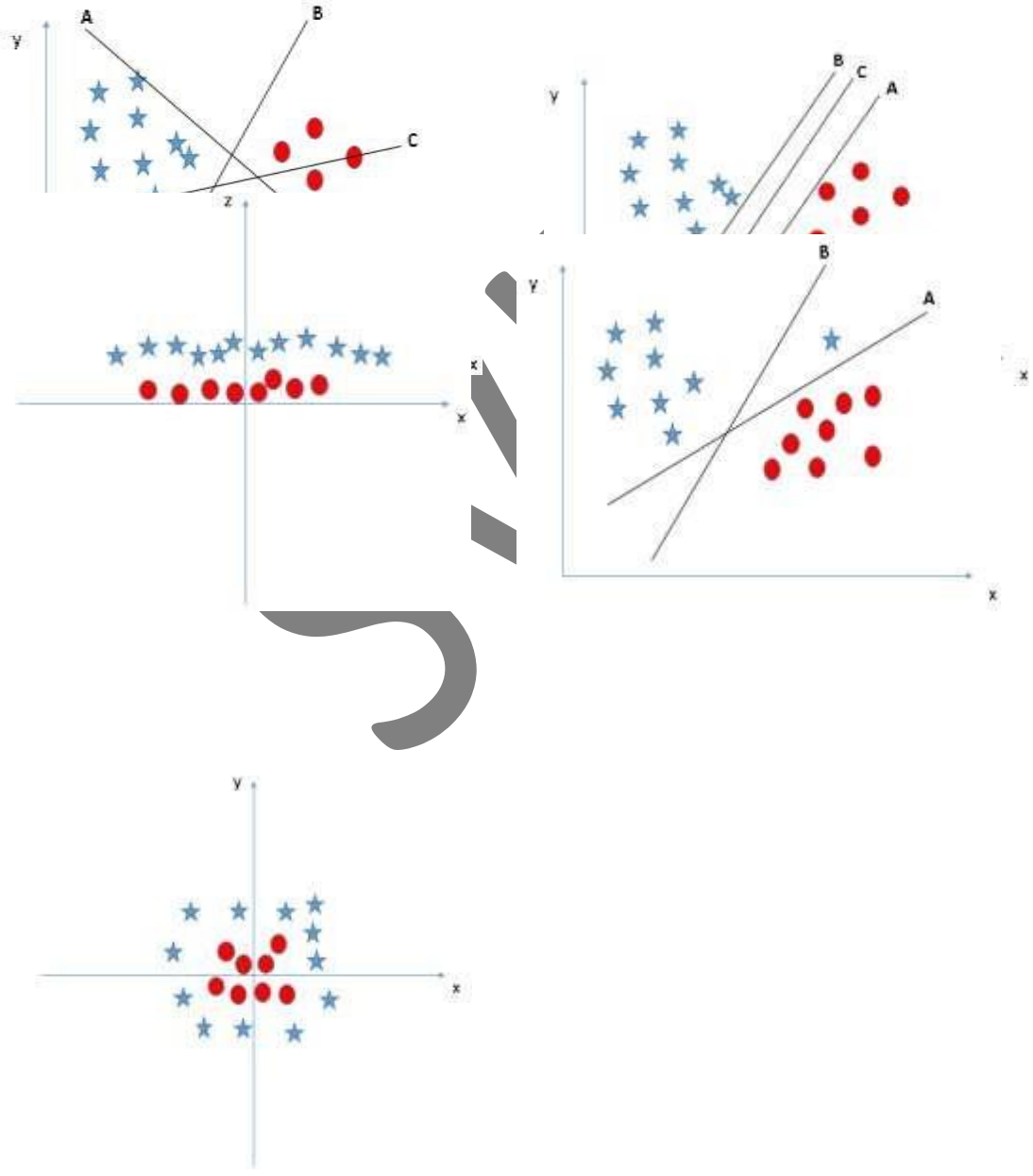
# Support Vector Machine

- "Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges.
- However, it is mostly used in classification problems.
- In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate.
- Then, we perform classification by finding the hyper-plane that differentiate the two classes very well (look at the below snapshot).

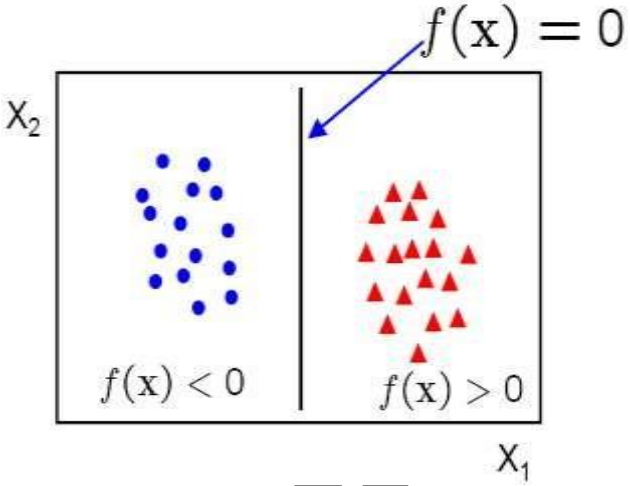


### How does it work?

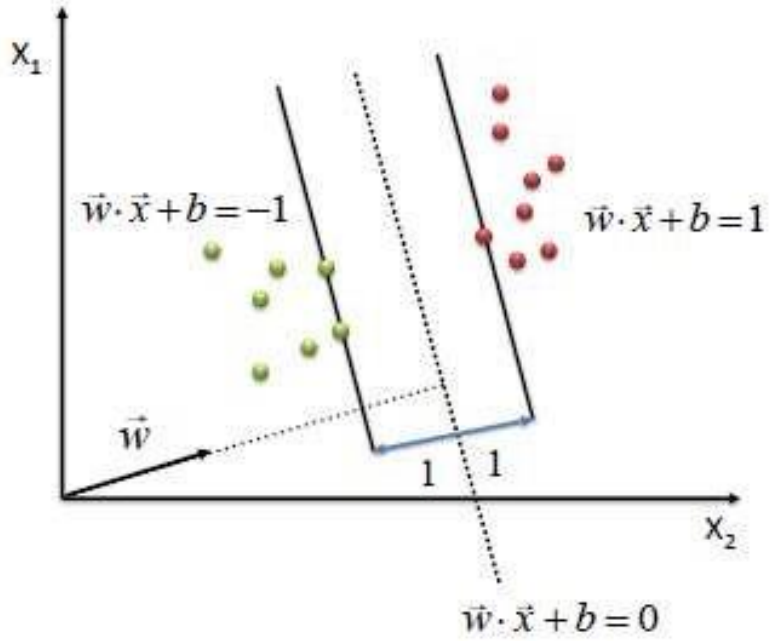
- Thumb rule to identify the right hyper-plane
- • Select the hyper-plane which segregates the two classes better
- • Maximizing the distances between nearest data point (either class) and hyper-plane. This distance is called as Margin.



## SVM Model




- f(x) = W.X + b
- W is the normal to the line, X is input vector and b the bias
- W is known as the weight vector




$$\max \frac{2}{\|w\|}$$

s.t.

$(w\cdot x+b)\geq 1, \forall x$ of class 1

$(w\cdot x+b)\leq -1, \forall x$ of class 2

## Advantages of SVM

- The main strength of SVM is that they work well even when the number of SVM features is much larger than the number of instances.
- It can work on datasets with huge feature space, such is the case in spam filtering, where a large number of words are the potential signifiers of a message being spam.

- Even when the optimal decision boundary is a nonlinear curve, the SVM transforms the variables to create new dimensions such that the representation of the classifier is a linear function of those transformed dimensions of the data.
- SVMs are conceptually easy to understand. They create an easy-to- understand linear classifier. By working on only a subset of relevant data,. they are computationally efficient. SVMs are now available with almost all data analytics toolsets.

## Disadvantages of SVM

The SVM technique has two major constraints

- It works well only with real numbers, i.e., all the data points in all the dimensions must be defined by numeric values only,
- It works only with binary classification problems. One can make a series of cascaded SVMs to get around this constraint.
- Training the SVMs is an inefficient and time consuming process, when the data is large.
- It does not work well when there is much noise in the data, and thus has to compute soft margins.
- The SVMs will also not provide a probability estimate of classification, i.e., the confidence level for classifying an instance.

# Module -1

## Introduction to Big Data

### 1.1 Need of Big Data

The rise in technology has led to the production and storage of voluminous amounts of data. Earlier megabytes ($10^6$ B) were used but nowadays petabytes ($10^{15}$ B) are used for processing, analysis, discovering new facts and generating new knowledge. Conventional systems for storage, processing and analysis pose challenges in large growth in volume of data, variety of data, various forms and formats, increasing complexity, faster generation of data and need of quickly processing, analyzing and usage.

Figure 1.1 shows data usage and growth. As size and complexity increase, the proportion of unstructured data types also increase.

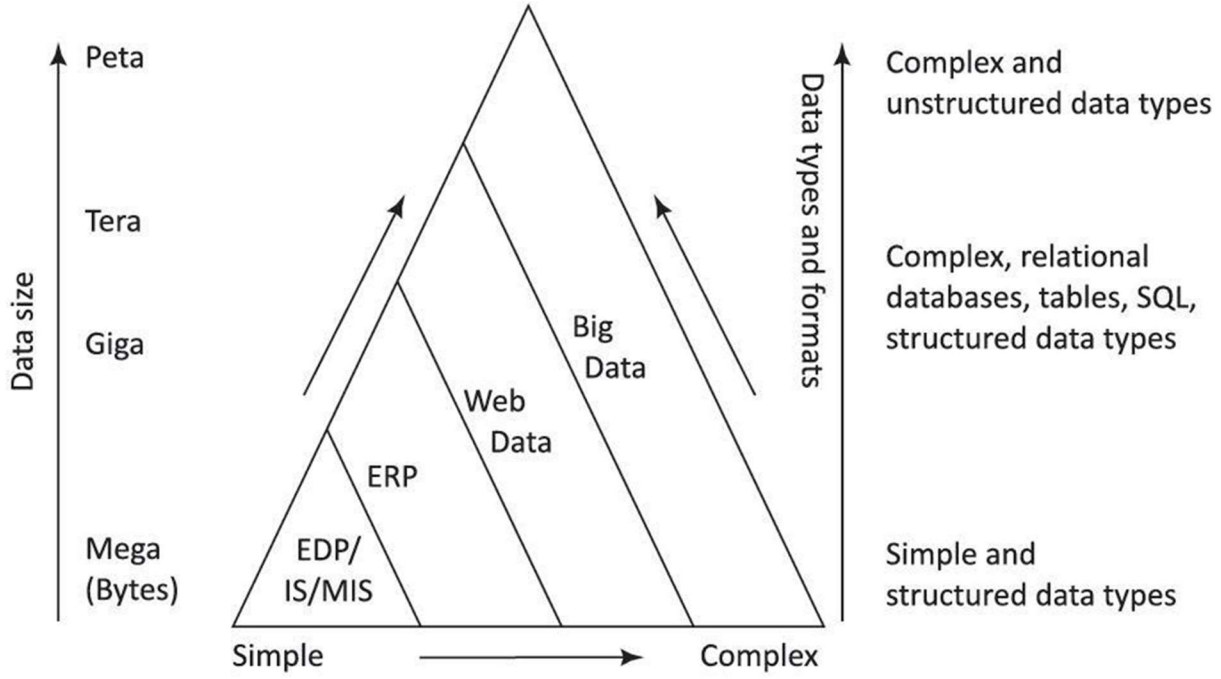



Figure 1.1 Evolution of Big Data and their characteristics

An example of a traditional tool for structured data storage and querying is RDBMS. Volume, velocity and variety (3Vs) of data need the usage of number of programs and tools for analyzing and processing at a very high speed.

## 1.2 BIG DATA

Data is information, usually in the form of facts or statistics that one can analyze or use for further calculations. Data is information that can be stored and used by a computer program. Data is information presented in numbers, letters, or other form. Data is information from series of observations, measurements or facts. Data is information from series of behavioral observations, measurements or facts.

### Web Data

*Web data* is the data present on web servers (or enterprise servers) in the form of text, images, videos, audios and multimedia files for web users. A user (client software) interacts with this data. A client can access (pull) data of responses from a server. The data can also publish (push) or post (after registering subscription) from a server. Internet applications including web sites, web services, web portals, online business applications, emails, chats, tweets and social networks provide and consume the web data.

- Examples of Web data
- Wikipedia,
- Google Maps,
- YouTube.
- Face Book

### 1.2.1 Classification of Data

Data can be classified as

- structured
- semi-structured
- multi-structured
- unstructured.

#### Structured Data

Structured data conform and associate with data schemas and data models. Structured data are found in tables (rows and columns). Nearly 15-20% data are in structured or semi-structured form.

Structured data enables the following:

- *data insert, delete, update and* append
- indexing to enable faster data retrieval
- *Scalability* which enables increasing or decreasing capacities and data processing operations such as, storing, processing and analytics
- *Transactions processing* which follows ACID rules (Atomicity, Consistency, Isolation and Durability)
- encryption and *decryption* for data security.

**Semi Structured Data**

Examples of semi-structured data are XML and JSON documents. Semi-structured data contain tags or other markers, which separate semantic elements and enforce hierarchies of records and fields within the data. Semi-structured form of data does not conform and associate with formal data model structures. Data do not associate data models, such as the relational database and table models.

**Multi Structured Data**

- Multi-structured data refers to data consisting of multiple formats of data, viz. structured, semi-structured and/or unstructured data.
- Multi-structured data sets can have many formats.
- They are found in non-transactional systems.
- For example, streaming data on customer interactions, data of multiple sensors, data at web or enterprise server or the data- warehouse data in multiple formats.

## *Unstructured Data*

- Data does not possess data features such as a table or a database.
- Unstructured data are found in file types such as .TXT, .CSV.

- Data may be as key-value pairs, such as hash key-value pairs.
- Data may have internal structures, such as in e- mails.
- The data do not reveal relationships, hierarchy relationships.
- The relationships, schema and features need to be separately established.

Examples of unstructured Data

- Mobile data: Text messages, chat messages, tweets, blogs and comments
- Website content data: YouTube videos, browsing data, e-payments, web store data, user-generated maps
- Social media data: For exchanging data in various forms
- Texts and documents
- Personal documents and e-mails
- Text internal to an organization: Text within documents, logs, survey results
- Satellite images, atmospheric data, surveillance, traffic videos, images from Instagram, Flickr (upload, access, organize, edit and share photos from any device from anywhere in the world).

## 1.2 Big Data Definitions

- Big Data is high-volume, high-velocity and/or high-variety information that requires new forms of processing for enhanced decision making, insight discovery and process optimization
- A collection of data sets so large or complex that traditional data processing applications are inadequate." -Wikipedia
- Data of a very large size, typically to the extent that its manipulation and management present significant logistical challenges-oxford English dictionary.
- Big Data refers to data sets whose size is beyond the ability of typical database software tool to capture, store, manage and analyze

### 1.2.2 Big Data Characteristics

- Volume: is related to size of the data
- Velocity: refers to the speed of generation of data.
- Variety: comprises of a variety of data
- Varacity: quality of data captured, which can vary greatly, affecting its accurate analysis

### 1.2.3 Big Data Types

- Social networks and web data, such as Facebook, Twitter, e-mails, blogs and YouTube.
- Transactions data and Business Processes {BPs) data, such as credit card transactions, flight bookings, etc. and public agencies data such as medical records, insurance business data etc.
- Customer master data such as data for facial recognition and for the name, date of birth, marriage anniversary, gender, location and income category,
- Machine-generated data, such as machine-to-machine or Internet of Things data, and the data from sensors, trackers, web logs and computer systems log.
- Computer generated data is also considered as machine generated data
- Human-generated data such as biometrics data, human—machine interaction data, e-mail records with a mail server and MySQL database of student grades.
- Humans also records their experiences in ways such as writing these in notebooks diaries, taking photographs or audio and video clips.
- Human-sourced information is now almost entirely digitized and stored everywhere from personal computers to social networks

### Examples of Big Data

- Chocolate Marketing Company with large number of installed Automatic Chocolate Vending Machines (ACVMs).
- Automotive Components and Predictive Automotive Maintenance Services

- (ACPAMS) rendering customer services for maintenance and servicing of (Internet) connected cars and its components
- Weather data Recording, Monitoring and Prediction (WRMP) Organization.

| Method | Type of Data |
|---|---|
| Data sources (traditional) | Data storage such as records, RDBMs, distributed databases, row-oriented In- memory data tables, column-oriented In-memory data tables, data warehouse, server, machine-generated data, human-sourced data, Business Process (BP) data, Business Intelligence (BI) data |
| Data formats (traditional) | Structured and semi-structured |
| Big Data sources | Data storage, distributed file system, Operational Data Store (ODS), data marts, data warehouse, NoSQL database (MongoDB, Cassandra), sensors data, audit trail of financial transactions, external data such as web, social media, weather data, health records |
| Big Data formats | Unstructured, semi-structured and multi-structured data |
| Data Stores structure | Web, enterprise or cloud servers, data warehouse, row-oriented data for OLTP, column-oriented for OLAP, records, graph database, hashed entries for key/value pairs |
| Processing data rates | Batch, near-time, real-time, streaming |
| Processing Big Data rates | High volume, velocity, variety and veracity, batch, near real-time and streaming data processing, |
| Analysis types | Batch, scheduled, near real-time datasets analytics |
| Big Data processin g methods | Batch processing (for example, using MapReduce, Hive or Pig), real-time processing (for example, using SparkStreaming, SparkSQL, Apache Drill) |
| Data analysis methods | Statistical analysis, predictive analysis, regression analysis, Mahout, machine learning algorithms, clustering algorithms, classifiers, text analysis, social network analysis, location-based analysis, diagnostic analysis, cognitive analysis |
| Data Usage | Human, business process, knowledge discovery, enterprise applications, Data |

## 1.2.5 Big Data Classification

Big Data can be classified on the basis of its characteristics that are used for designing data architecture for processing and analytics.

### 1.2.6 Big Data Handling Techniques

Following are the techniques deployed for Big Data storage, applications, data management and mining and analytics:

- Huge data volumes storage, data distribution, high-speed networks and high-performance computing
- Applications scheduling using open source, reliable, scalable, distributed file system, distributed database, parallel and distributed computing systems, such as Hadoop or Spark
- Open source tools which are scalable, elastic and provide virtualized environment, clusters of data nodes, task and thread management
- Data management using NoSQL, document database, column-oriented database, graph database and other form of databases used as per needs of the applications and in-memory data management using columnar or Parquet formats during program execution
- Data mining and analytics, data retrieval, data reporting, data visualization and machine-learning Big Data tools.

## 1.3 Scalability and Parallel Processing

- Big Data needs processing of large data volume, and therefore needs intensive computations.
- Processing complex applications with large datasets (terabyte to petabyte datasets) need hundreds of computing nodes.
- Processing of this much distributed data within a short time and at minimum cost is problematic.
- Scalability is the capability of a system to handle the workload as per the magnitude of the work.
- System capability needs increment with the increased workloads.
- When the workload and complexity exceed the system capacity, scale it up and scale it out.
- Scalability enables increase or decrease in the capacity of data storage, processing&

analytics.

### 1.3.1 Analytical Scalability

***Vertical scalability*** means scaling up the given system's resources and increasing the system's analytics, reporting and visualization capabilities. This is an additional way to solve problems of greater complexities. Scaling up means designing the algorithm according to the architecture that uses resources efficiently.

x terabyte of data take time t for processing, code size with increasing complexity increase by factor n, then scaling up means that processing takes equal, less or much less than (n * t).

Horizontal scalability means increasing the number of systems working in coherence and scaling out the workload. Processing different datasets of a large dataset deploys horizontal scalability. Scaling out means using more resources and distributing the processing and storage tasks in parallel. The easiest way to scale up and scale out execution of analytics software is to implement it on a bigger machine with more CPUs for greater volume, velocity, variety and complexity of data. The software will definitely perform better on a bigger machine.

### 1.3.2 Massive Parallel Processing Platforms

Parallelization of tasks can be done at several levels:

- distributing separate tasks onto separate threads on the same CPU
- distributing separate tasks onto separate CPUs on the same computer
- distributing separate tasks onto separate computers.

### 1.3.3 Distributed Computing Model

A distributed computing model uses cloud, grid or clusters, which process and analyze big and large datasets on distributed computing nodes connected by high-speed networks.

Big Data processing uses a parallel, scalable and no-sharing program model, such as MapReduce, for computations on it.

| Distributed Computing on multiple nodes | Big data | Large data | Small to Medium data |
|---|---|---|---|
| Distributed computing | Yes | Yes | No |
| Parallel computing | Yes | Yes | No |
| Scalable computing | Yes | Yes | No |
| Shared nothing (No in-between data sharing and inter-processor communication) | ves | Limited sharing | No |
| Shared in-between between the distributed nodes/clusters | No | Limited sharing | Yes |

### 1.3.4 Cloud Computing

- "Cloud computing is a type of Internet-based computing that provides shared processing resources and data to the computers and other devices on demand."
- One of the best approach for data processing is to perform parallel and distributed computing in a cloud-computing environment
- Cloud resources can be Amazon Web Service (AWS) Elastic Compute Cloud (EC2), Microsoft Azure or Apache CloudStack.

### Features of Cloud Computing

- on-demand service
- resource pooling,
- scalability,
- accountability,
- broad network access.
- Cloud services can be accessed from anywhere and at any time through the Internet.

## Cloud Services

There are three types of Cloud Services

- Infrastructure as a Service (IaaS):
- Platform as a Service (PaaS):
- Software as a Service (SaaS):

## Infrastructure as a Service (IaaS):

- Providing access to resources, such as hard disks, network connections, databases storage, data center and virtual server spaces is Infrastructure as a Service (IaaS).
- Some examples are Tata Communications, Amazon data centers and virtual servers.
- Apache CloudStack is an open source software for deploying and managing a large network of virtual machines, and offers public cloud services which provide highly scalable Infrastructure as a Service (IaaS).

## Platform as a Service

- It implies providing the runtime environment to allow developers to build applications and services, which means cloud Platform as a Service.
- Software at the clouds support and manage the services, storage, networking, deploying, testing, collaborating, hosting and maintaining applications.
- Examples are Hadoop Cloud Service (IBM BigInsight, Microsoft Azure HD Insights, Oracle Big Data Cloud Services).

## Software as a service

- Providing software applications as a service to end- users is known as Software as a Service.
- Software applications are hosted by a service provider and made available to customers over the Internet.

- Some examples are SQL Google SQL, IBM BigSQL, Microsoft Polybase and Oracle Big Data SQL.

### 1.3.5 Grid Computing

- Grid Computing refers to distributed computing, in which a group of computers from several locations are connected with each other to achieve a common task.
- The computer resources are heterogeneously and geographically dispersing.
- A group of computers that might spread over remotely comprise a grid.
- A single grid of course, dedicates at an instance to a particular application only.
- Grid computing, similar to cloud computing, is scalable.
- Cloud computing depends on sharing of resources (for example, networks, servers, storage, applications and services) to attain coordination and coherence among resources similar to grid computing.
- Similarly, grid also forms a distributed network for resource integration.

### Cluster Computing

A cluster is a group of computers connected by a network. The group works together to accomplish the same task. Clusters are used mainly for load balancing. They shift processes between nodes to keep an even load on the group of connected computers.

| Distributed computing | Cluster computing | Grid computing |
|---|---|---|
| • Loosely coupled<br>• Heterogeneous<br>• Single administration | • Tightly coupled<br>• Homogeneous<br>• Cooperative working | • Large scale<br>• Cross organizational<br>• Geographical distribution<br>• Distributed management |

### 1.3.6 Volunteer Computing

Volunteers provide computing resources to projects of importance that use resources to do distributed computing and/or storage. Volunteer computing is a distributed computing paradigm which uses computing resources of the volunteers. Volunteers are organizations or members who own personal computers. Projects examples are science-related projects executed by universities or academia in general.

Some issues with volunteer computing systems are:

- Volunteered computers heterogeneity
- Drop outs from the network over time
- Their sporadic availability

## 1.4 Designing the Data Architecture

Big Data architecture is the logical and/or physical layout/structure of how Big Data will be stored, accessed and managed within a Big Data or IT environment. Architecture logically defines how Big Data solution will work, the core components (hardware, database, software, storage) used, flow of information, security and more.

Data analytics need the number of sequential steps. Big Data architecture design task simplifies when using the logical layers approach. Figure 1.2 shows the logical layers and the functions which are considered in Big Data architecture

### Data processing architecture consists of five layers:

- identification of data sources,
- acquisition, ingestion, extraction, pre-processing, transformation of data,
- Data storage at files, servers, cluster or cloud,
- data-processing,
- data consumption in the number of programs and tools.

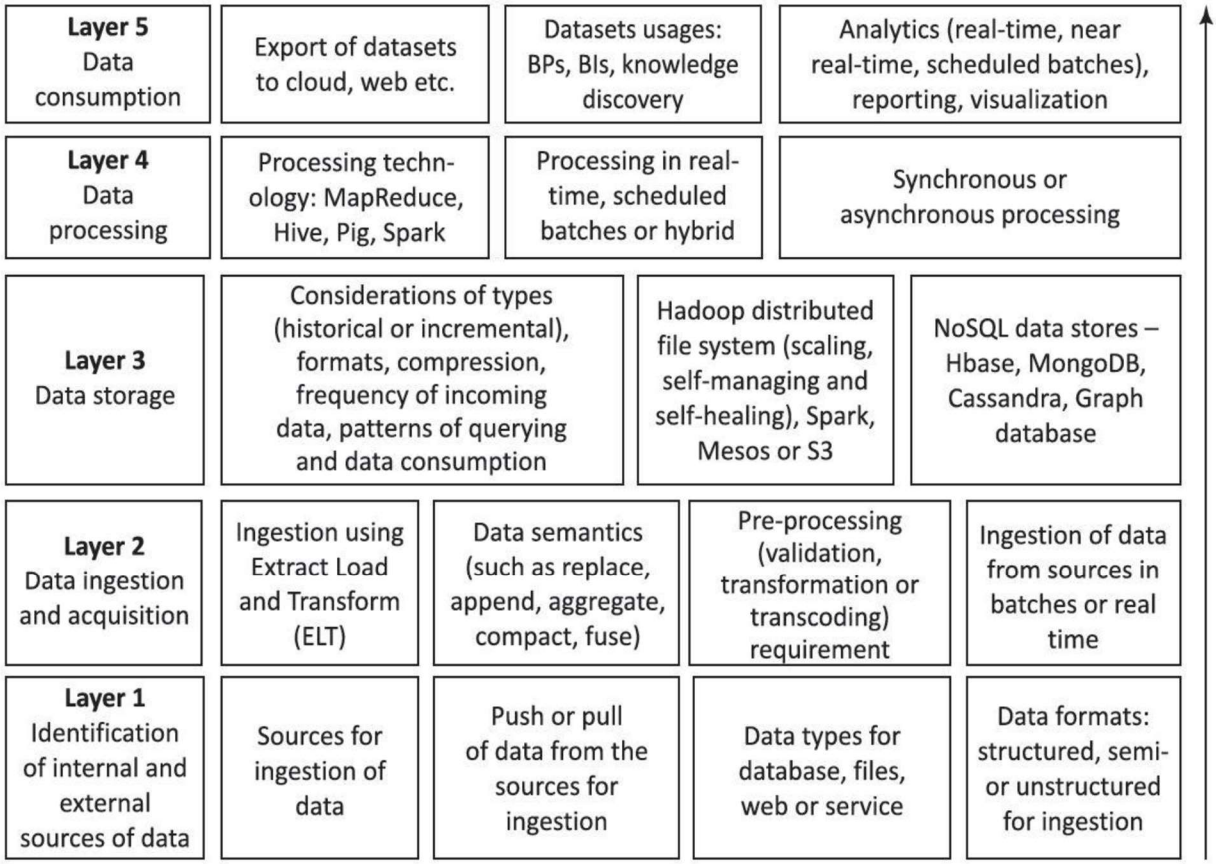



Figure 1.2 Design of logical layers in a data processing architecture, and functions in the layers

Logical layer 1 (L1) is for identifying data sources, which are external, internal or both. The layer 2 (L2) is for data-ingestion.Data ingestion means a process of absorbing information, just like the process of absorbing nutrients and medications into the body by eating or drinking them .Ingestion is the process of obtaining and importing data for immediate use or transfer. Ingestion may be in batches or in real time using pre-processing or semantics.

**Layer 1**

- L1 considers the following aspects in a design:
  - Amount of data needed at ingestion layer 2 (L2)

- Push from L1 or pull by L2 as per the mechanism for the usages
- Source data-types: Database, files, web or service

- Source formats, i.e., semi-structured, unstructured or structured.

**Layer 2**

- Ingestion and ETL processes either in real time, which means store and use the data as generated, or in batches.
- Batch processing is using discrete datasets at scheduled or periodic intervals of time.

**Layer 3**

- Data storage type (historical or incremental), format, compression, incoming data
- frequency, querying patterns and consumption requirements for L4 or L5
- Data storage using Hadoop distributed file system or NoSQL data stores—HBase, Cassandra, MongoDB.

**Layer 4**

- Data processing software such as MapReduce, Hive, Pig, Spark, Spark Mahout, Spark Streaming
- Processing in scheduled batches or real time or hybrid
- Processing as per synchronous or asynchronous processing requirements at L5.

**Layer 5**

- Data integration
- Datasets usages for reporting and visualization
- Analytics (real time, near real time, scheduled batches), BPs, BIs, knowledge discovery
- Export of datasets to cloud, web or other systems

### 1.4.2 Managing Data for Analysis

Data managing means enabling, controlling, protecting, delivering and enhancing the value of data and information asset. Reports, analysis and visualizations need well- defined data.

Data management functions include:

1. Data assets creation, maintenance and protection
2. Data governance, which includes establishing the processes for ensuring the availability, usability, integrity, security and high-quality of data. The processes enable trustworthy data availability for analytics, followed by the decision making at the enterprise.
3. Data architecture creation, modelling and analysis
4. Database maintenance, administration and management system. For example, RDBMS (relational database management system), NoSQL
5. Managing data security, data access control, deletion, privacy and security
6. Managing the data quality
7. Data collection using the ETL process
8. Managing documents, records and contents
9. Creation of reference and master data, and data control and supervision
10. Data and application integration
11. Integrated data management, enterprise-ready data creation, fast access and analysis, automation and simplification of operations on the data,
12. Data warehouse management
13. Maintenance of business intelligence
14. Data mining and analytics algorithms.

## 1.5 Data Source

Applications, programs and tools use data. Sources can be external, such as sensors, trackers, web logs, computer systems logs and feeds. Sources can be machines, which source data from data-creating programs.

A source can be internal. Sources can be data repositories, such as database, relational database, flat file, spreadsheet, mail server, web server, directory services, even text or files such as comma-separated values (CSV) files. Source may be a data store for applications

### 1.5.1 Types of Data Source

- structured
- semi-structured
- multi-structured or unstructured

**Structured Data Source**

- Data source for ingestion, storage and processing can be a file, database or streaming data.
- The source may be on the same computer running a program or a networked computer
- Structured data sources are SQL Server, MySQL, Microsoft Access database, Oracle DBMS, IBM DB2, Informix, Amazon SimpleDB or a file-collection directory at a server.

**Unstructured Data Source**

- Unstructured data sources are distributed over high-speed networks.
- The data need high velocity processing. Sources are from distributed file systems.
- The sources are of file types, such as .txt (text file), .csv (comma separated values file). Data may be as key value pairs, such as hash key-values pairs

**Data Sources - Sensors, Signals and GPS**

The data sources can be sensors, sensor networks, signals from machines, devices,

controllers and intelligent edge nodes of different types in the industry M2M communication and the GPS systems.

Sensors are electronic devices that sense the physical environment. Sensors are devices which are used for measuring temperature, pressure, humidity, light intensity, traffic in proximity, acceleration, locations, object(s) proximity, orientations and magnetic intensity, and other physical states and parameters. Sensors play an active role in the automotive industry.

RFIDs and their sensors play an active role in RFID based supply chain management, and tracking parcels, goods and delivery.

Sensors embedded in processors, which include machine-learning instructions, and wireless communication capabilities are innovations. They are sources in IoT applications.

## 1.5.2 Data Quality

High quality means data, which enables all the required operations, analysis, decisions, planning and knowledge discovery correctly. Five R's as follows:

- Relevancy,
- recency,
- range,
- robustness
- reliability.

**Data Integrity**

Data integrity refers to the maintenance of consistency and accuracy in data over its usable life. Software, which store, process, or retrieve the data, should maintain the integrity of data. Data should be incorruptible

**Factors Affecting Data Quality**

- Data Noise

- Outlier
- Missing Value
- Duplicate value

**Data Noise**

- Noise One of the factors effecting data quality is noise.
- Noise in data refers to data giving additional meaningless information besides true (actual/required) information.
- Noise is random in character, which means frequency with which it occurs is variable over time**.**

**Outlier**

- An *outlier* in data refers to data, which appears to not belong to the dataset.For example, data that is outside an expected range.
- Actual outliers need to be removed from the dataset, else the result will be effected by a small or large amount.

**Missing Value, duplicate Value**

- Missing Values Another factor effecting data quality is missing values. Missing value implies data not appearing in the data set.
- Duplicate Values Another factor effecting data quality is duplicate values. Duplicate value implies the same data appearing two or more times in a dataset.

### 1.5.3 Data Preprocessing

Data pre-processing is an important step at the ingestion layer.Pre-processing is a must before data mining and analytics. Pre-processing is also a must before running a Machine Learning (ML) algorithm.Pre-processing needs are:

- Dropping out of range, inconsistent and outlier values
- Filtering unreliable, irrelevant and redundant information

- Data cleaning, editing, reduction and/or wrangling
- Data validation, transformation or transcoding
- ELT processing

**Data Cleaning**

- *Data cleaning* refers to the process of removing or correcting incomplete, incorrect, inaccurate or irrelevant parts of the data after detecting them.
- Data cleaning is done before mining of data. Incomplete or irrelevant data may result into misleading decisions.
- Data cleaning tools help in refining and structuring data into usable data. Examples of such tools are OpenRefine and DataCleaner.

**Data Enrichment**

- "Data enrichment refers to operations or processes which refine, enhance or improve the raw data.“
- Data editing refers to the process of reviewing and adjusting the acquired datasets.
- The editing controls the data quality.
- Editing methods are (i) interactive, (ii) selective, (iii) automatic, (iv) aggregating and (v) distribution.

**Data Reduction**

- Data reduction enables the transformation of acquired information into an ordered, correct and simplified form.
- Data wrangling refers to the process of transforming and mapping the data. Results from analytics are then appropriate and valuable.
- mapping enables data into another format, which makes it valuable for analytics and data visualizations

Data format using preprocessing

- Comma-separated values CSV
- Java Script Object Notation (JSON) as batches of object arrays or resource arrays
- Tag Length Value (TLV)
- Key-value pairs
- Hash-key-value pair

### 1.5.4 Data Export to Cloud

Figure 1.3 shows resulting data pre-processing, data mining, analysis, visualization and data store. The data exports to cloud services. The results integrate at the enterprise server or data warehouse.

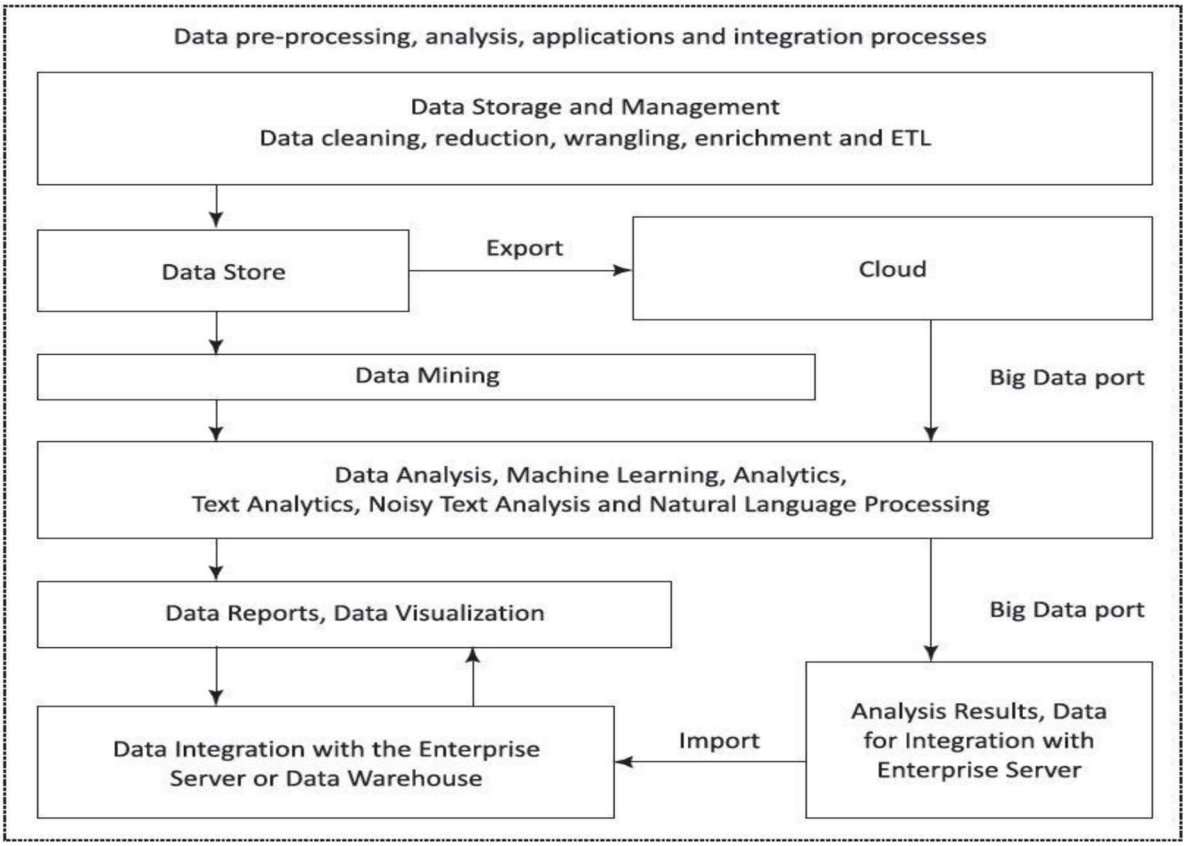



**Figure 1.3** Data pre-processing, analysis, visualization, data store export

**Cloud Services**

Cloud offers various services. These services can be accessed through a cloud client (client

application), such as a web browser, SQL or other client. Figure 1.4 shows data-store export from machines, files, computers, web servers and web services. The data exports to clouds, such as IBM, Microsoft, Oracle, Amazon, Rackspace, TCS, Tata Communications or Hadoop cloud services.

**Figure 1.4 Data store export from machines, files, computers, web servers and web services**

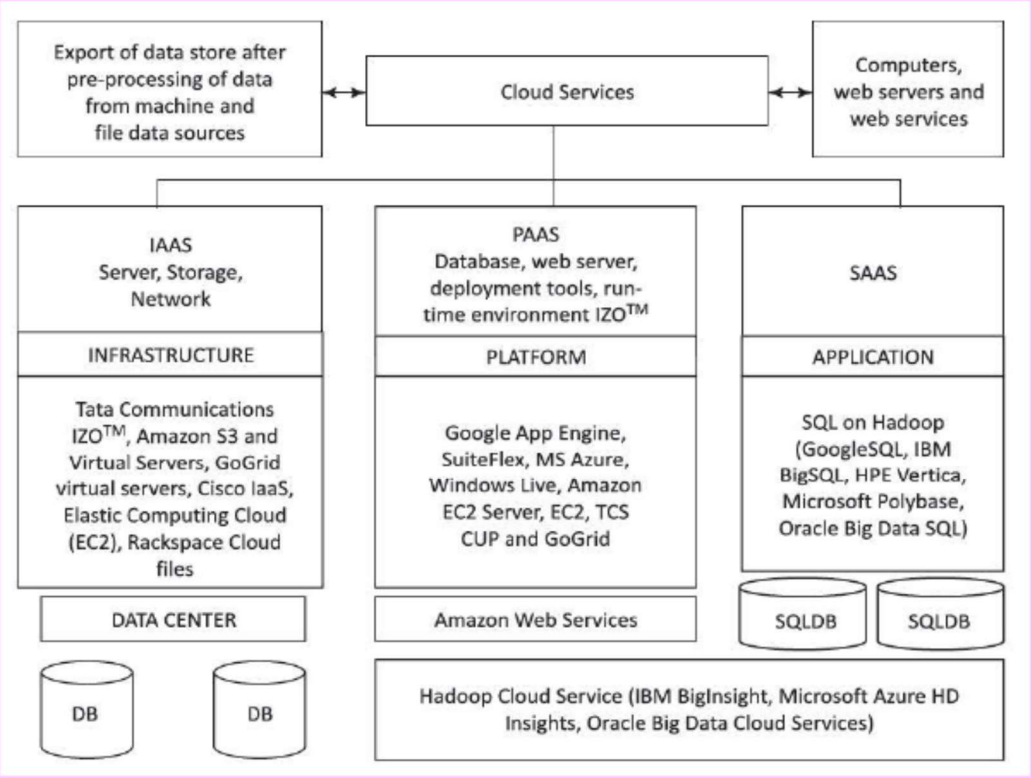



Figure 1.4 Data store export from machines, files, computers, web servers and web services

**Export of Data to AWS and Rackspace Clouds**

Google cloud platform provides a cloud service called BigQuery Figure 1.5 shows BigQuery cloud service at Google cloud platform. The data exports from a table or partition schema,JSON, CSV or AVRO files from data sources after the pre-processing.

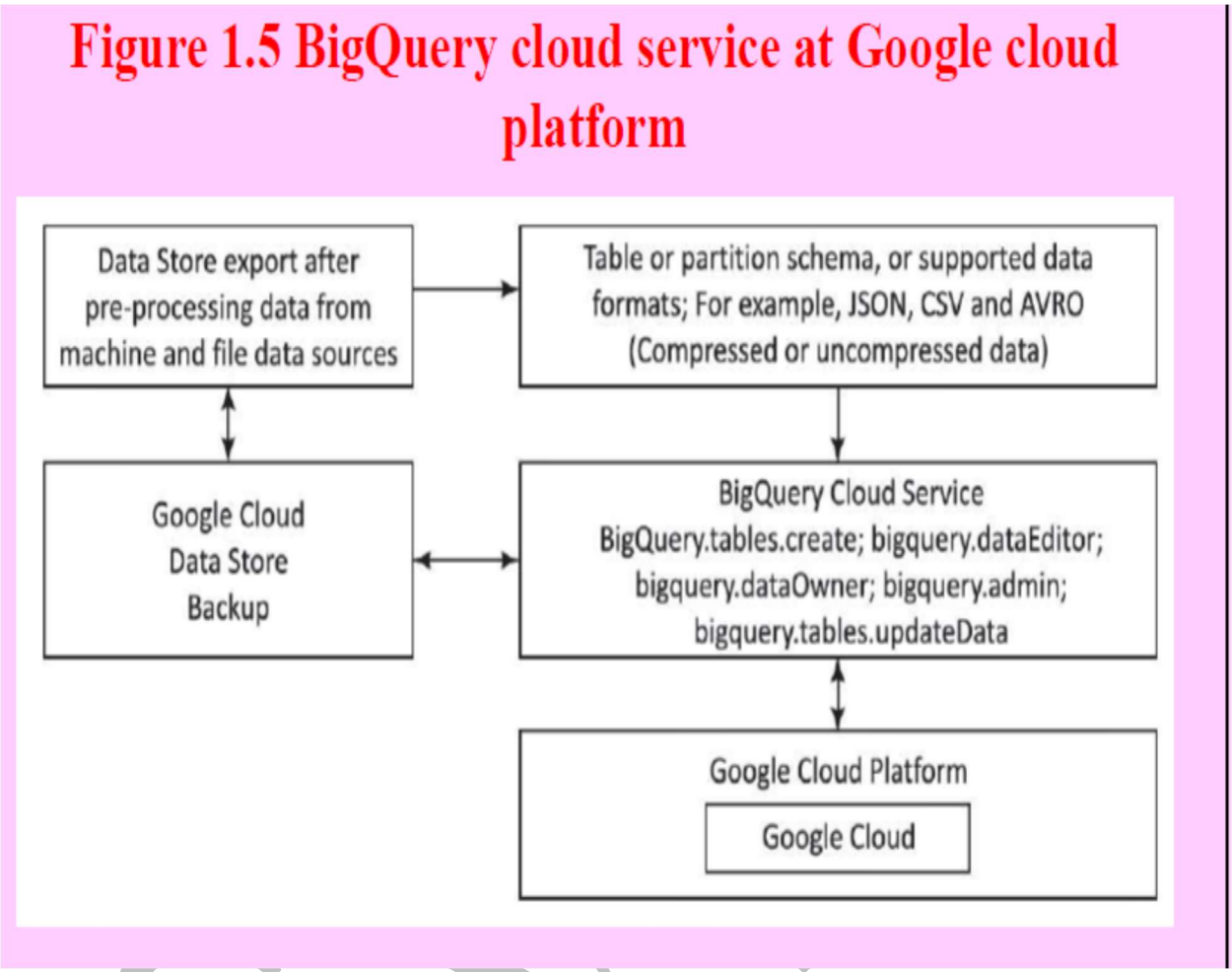



**Figure 1.5** BigQuery cloud service at Google cloud platform

Data Store first pre-processes from machine and file data sources. Pre-processing transforms the data in table or partition schema or supported data formats. For example, JSON, CSV and AVRO. Data then exports in compressed or uncompressed data formats.

Cloud service BigQuery consists of bigquery.tables.create; bigquery.dataEditor; bigquery.dataOwner; bigquery.admin; bigquery.tables.updateData and other service functions. Analytics uses Google Analytics 360. BigQuery cloud exports data to a Google cloud or cloud backup only.

## 1.6 Data Storage and Analysis

This section describe data storage and analysis, and comparison between Big Data management and analysis with traditional database management systems.

### 1.6.1 Data Storage and Management: Traditional Systems

- Traditional systems use structured or semi-structured data
- The sources of structured data store are:
- Traditional relational database-management system (RDBMS) data, such as MySQL DB2, enterprise server and data warehouse

**SQL**

An RDBMS uses SQL (Structured Query Language). SQL is a language for viewing or changing (update, insert or append or delete) databases.

1. *Create schema, Create schema,* which is a structure which contains description of objects (base tables, views, constraints) created by a user. The user can describe the data and define the data in the database.
2. *Create catalog,* which consists of a set of schemas which describe the database.
3. *Data Definition Language* (DDL) for the commands which depicts a database, that include creating, altering and dropping of tables and establishing the constraints.A user can create and drop databases and tables, establish foreign keys, create view, stored procedure, functions in the database etc.
4. *Data Manipulation Language* (DML) for commands that maintain and query the database. A user can manipulate (INSERT/UPDATE) and access (SELECT) the data.
5. *Data Control Language* (DCL) for commands that control a database, and include administering of privileges and committing. A user can set (grant, add or revoke) permissions on tables, procedures and views.

***Distributed Database Management System***

- A distributed DBMS (DDBMS) is a collection of logically interrelated databases at multiple system over a computer network.
- A collection of logically related databases.

- Cooperation between databases in a transparent manner.
- be 'location independent' which means the user is unaware of where the data is located, and it is possible to move the data from one physical location to another without affecting the user.

**In-Memory Column Formats Data**

- A columnar format in-memory allows faster data retrieval when only a few columns in a table need to be selected during query processing or aggregation.
- *Online Analytical Processing* (OLAP) in real-time transaction processing is fast when using in-memory column format tables.
- *The CPU accesses all columns in a single instance of access to the memory in columnar format in memory data-storage.*

***In-Memory* Row *Format Databases***

- A row format in-memory allows much faster data processing during OLTP
- Each row record has corresponding values in multiple columns and the on-line values store at the consecutive memory addresses in row format.

***Enterprise Data-Store Server and Data Warehouse***

- Enterprise data server use data from several distributed sources which store data using various technologies.
- All data *merge* using an integration tool.
- Integration enables collective viewing of the datasets at the data warehouse.
- Enterprise data integration may also include integration with application(s), such as analytics, visualization, reporting, business intelligence and knowledge discovery

Following are some standardised business processes, as defined in the Oracle application-integration architecture:

- Integrating and enhancing the existing systems and processes
- Business intelligence
- Data security and integrity

- New business services/products (Web services)
- Collaboration/knowledge management
- Enterprise architecture/SOA
- e-commerce
- External customer services
- Supply chain automation/visualization
- Data centre optimization

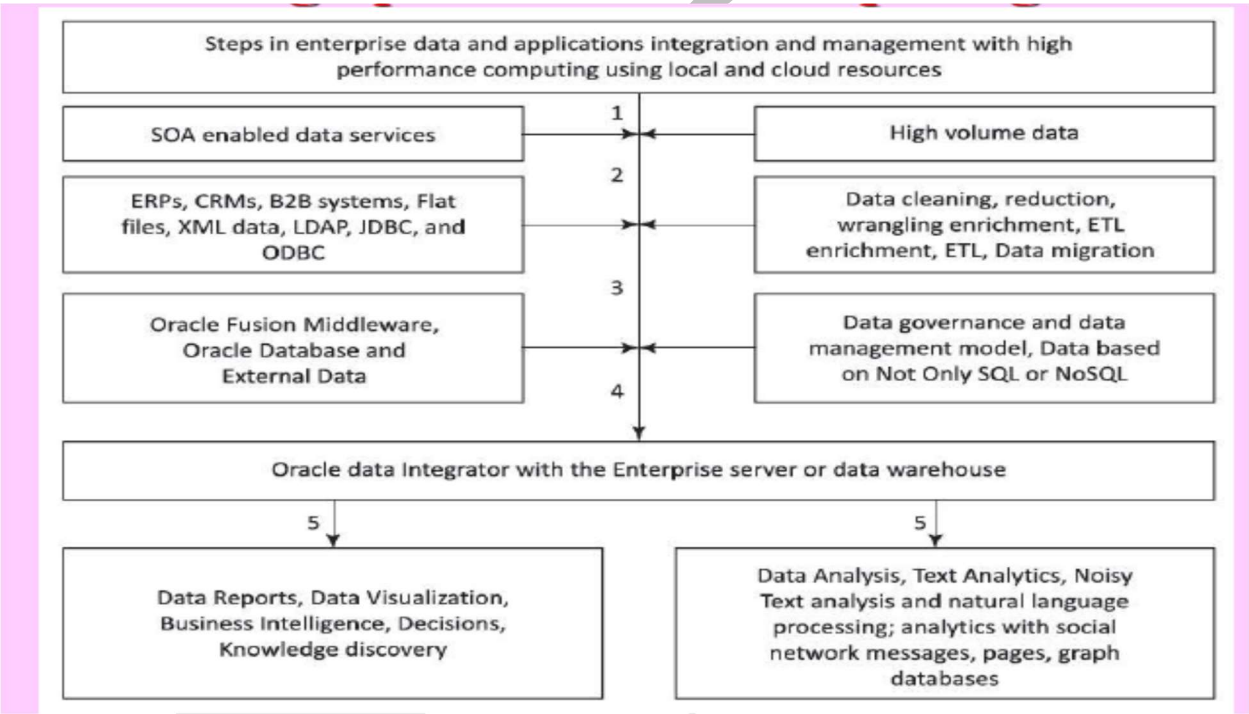



Figure 1.6 Steps 1 to 5 in Enterprise data integration and management with Big- Data for high performance computing using local and cloud resources for the analytics, applications and services

## 1.6.2 Big Data Storage

### NO SQL

- NoSQL databases are considered as semi-structured data. Big Data Store uses NoSQL. NOSQL stands for No SQL or Not Only SQL.
- The stores do not integrate with applications using SQL. NoSQL is also used in cloud data

store.

- Features ofNoSQL are as follows:
- It is a class of non-relational data storage systems, and the flexible data models and multiple schema:
- Class consisting of uninterrupted key/value or big hash table
- Class consisting of unordered keys and using JSON (PNUTS)
- Class consisting of ordered keys and semi-structured data storage systems [BigTable, Cassandra (used in Facebook/Apache) and HBase]
- Do not use the JOINS
- Data written at one node can replicate at multiple nodes, therefore Data storage is fault-tolerant,
- May relax the ACID rules during the Data Store transactions.

**Table 1.4** Various data sources and examples of usages and tools

| Data Source | Examples of Usages | Example of Tools |
|---|---|---|
| Relational databases | Managing business applications involving structured data | Microsoft Access, Oracle, IBM DB2, SQL Server, MySQL, PostgreSQL Composite, SQL on Hadoop [HPE (Hewlett Packard Enterprise) Vertica, IBM BigSQL, Microsoft Polybase, Oracle Big Data SQL] |
| Analysis databases (MPP, columnar, In-memory) | High performance queries and analytics | Sybase IQ, Kognitio, Terradata, Netezza, Vertica, ParAccel, ParStream, Infobright, Vectorwise, |
| NoSQL databases (Key-value pairs, Columnar format, documents, | Key-value pairs, fast read/write using collections of name-value pairs for storing any type of data; Columnar format, documents, | Key-value pair databases: Riak DS (Data Store), OrientDB, Column format databases (HBase, Cassandra), Document oriented databases: CouchDB, MongoDB; Graph |

| | | |
|---|---|---|
| Objects, graph) | objects, graph DBs and DSs | databases (Neo4j, Tetan) |
| Hadoop clusters | Ability to process large data sets across a distributed computing environment | Cloudera, Apache HDFS |
| Web applications | Access to data generated from web applications | Google Analytics, Twitter |
| Cloud data | Elastic scalable outsourced databases, and data administration services | Amazon Web Services, Rackspace, GoogleSQL |
| Individual data | Individual productivity | MS Excel, CSV, TLV, JSON, MIME type |
| Multidimensional | Well-defined bounded exploration especially popular for financial applications | Microsoft SQL Server Analysis Services |
| Social media data | Text data, images, videos | Twitter, LinkedIn |

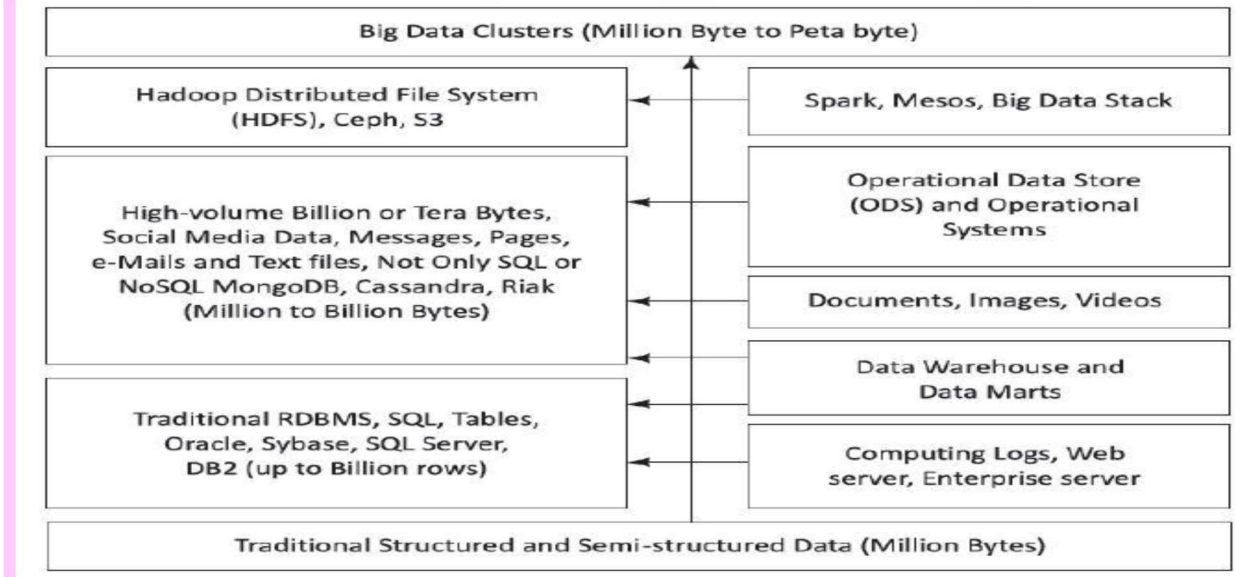



**Figure 1.7** Coexistence ofRDBMS for traditional server data, NoSQL and Hadoop, Spark and compatible Big Data Clusters

### 1.6.3 Big Data Platform

A Big Data platform supports large datasets and volume of data. The data generate at a higher velocity, in more varieties or in higher veracity. Managing Big Data requires large resources of MPPs, cloud, parallel processing and specialized tools. Bigdata platform should provision tools and services for:

1. storage, processing and analytics,
2. developing, deploying, operating and managing Big Data environment,
3. reducing the complexity of multiple data sources and integration of applications into one cohesive solution,
4. custom development, querying and integration with other systems, and
5. the traditional as well as Big Data techniques.

Data management, storage and analytics of Big data captured at the companies and services require the following:

1. New innovative non-traditional methods of storage, processing and analytics
2. Distributed Data Stores
3. Creating scalable as well as elastic virtualized platform (cloud computing)
4. Huge volume of Data Stores
5. Massive parallelism
6. High speed networks
7. High performance processing, optimization and tuning
8. Data management model based on Not Only SQL or NoSQL
9. In-memory data column-formats transactions processing or *dual in-memory data* columns as well as row formats for OLAP and OLTP
10. Data retrieval, mining, reporting, visualization and analytics

11. Graph databases to enable analytics with social network messages, pages and dataanalytics
12. Machine learning or other approaches
13. Big data sources: Data storages, data warehouse, Oracle Big Data, MongoDB NoSQL,Cassandra NoSQL
14. Data sources: Sensors, Audit trail of Financial transactions data, external data suchas Web, Social Media, weather data, health records data.

**Hadoop**

Big Data platform consists of Big Data storage(s), server(s) and data management and business intelligence software. Storage can deploy Hadoop Distributed File System (HDFS), NoSQL data stores, such as HBase, MongoDB, Cassandra. HDFS system is an open source storage system. HDFS is a scaling, self-managing and self-healing file system.

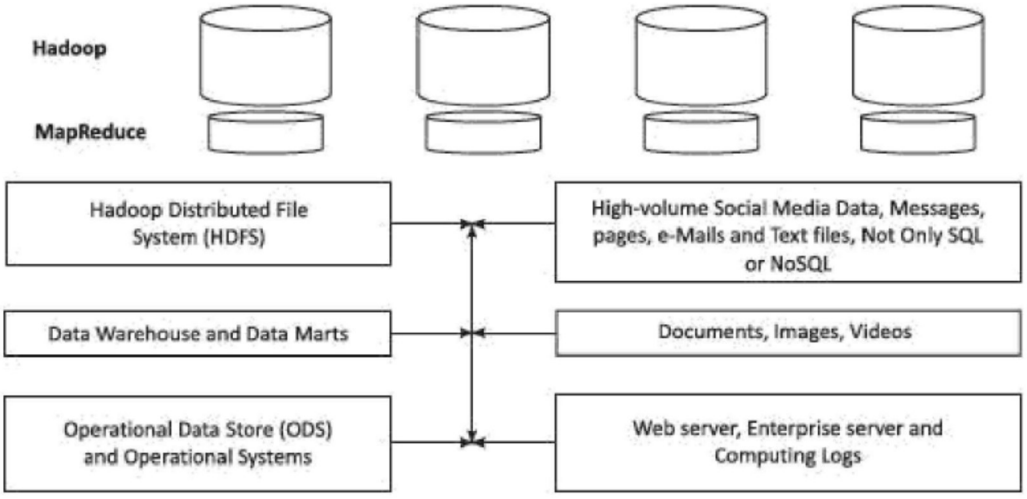



**Figure 1.8** Hadoop based Big Data environment

**Figure 1.8** Hadoop based Big Data environment

The Hadoop system packages application-programming model. Hadoop is a scalable and reliable parallel computing platform. Hadoop manages Big Data distributed databases. Figure 1.8 shows Hadoop based Big Data environment. Small height cylinders represent MapReduce and big ones represent the Hadoop.

### Big Data Stack

A stack consists of a set of software components and data store units. Applications, machine-learning algorithms, analytics and visualization tools use Big Data Stack (BDS) at a cloud service, such as Amazon EC2, Azure or private cloud. The stack uses cluster of high performance machines.

| Types | Examples |
|---|---|
| MapReduce | Hadoop, Apache Hive, Apache Pig, Cascading, Cascalog, mrjob (Python MapReduce library), Apache S4, MapR, Apple Acunu, Apache Flume, Apache Kafka |
| NoSQL Databases | MongoDB, Apache CouchDB, Apache Cassandra, Aerospike, Apache HBase, Hypertable |
| Processing | Spark, IBM BigSheets, PySpark, R, Yahoo! Pipes, Amazon Mechanical Turk, Datameer, Apache Solr/Lucene, ElasticSearch |
| Servers | Amazon ECZ, S3, GoogleQuery, Google App Engine, AWS Elastic Beanstalk, Salesforce Heroku |
| Storage | Hadoop Distributed File System, Amazon S3, Mesos |

**Table 1.5** Tools for Big Data environment

## 1.6.2 Big Data Analytics

Data Analytics can be formally defined as the statistical and mathematical data analysis that clusters, segments, ranks and predicts future possibilities. An important feature of data analytics is its predictive, forecasting and prescriptive capability. Analytics uses historical data and forecasts new values or results. Analytics suggests techniques which will provide the most efficient and beneficial results for an enterprise

Analysis of data is a process of inspecting, cleaning, transforming and modeling data with the goal of discovering useful information, suggesting conclusions and supporting decision making

### Phases in analytics

Analytics has the following phases before deriving the new facts, providing business intelligence and generating new knowledge.

1. *Descriptive analytics* enables deriving the additional value from visualizations

andreports

2. *Predictive analytics* is advanced analytics which enables extraction of new facts andknowledge, and then predicts/forecasts
3. *Prescriptive analytics* enable derivation of the additional value and undertake betterdecisions for new option(s) to maximize the profits
4. *Cognitive analytics* enables derivation of the additional value and undertake better decision.

Figure 1.9 shows an overview of a reference model for analytics architecture. The figure also shows on the right-hand side the Big Data file systems, machine learning algorithmsand query languages and usage of the Hadoop ecosystem

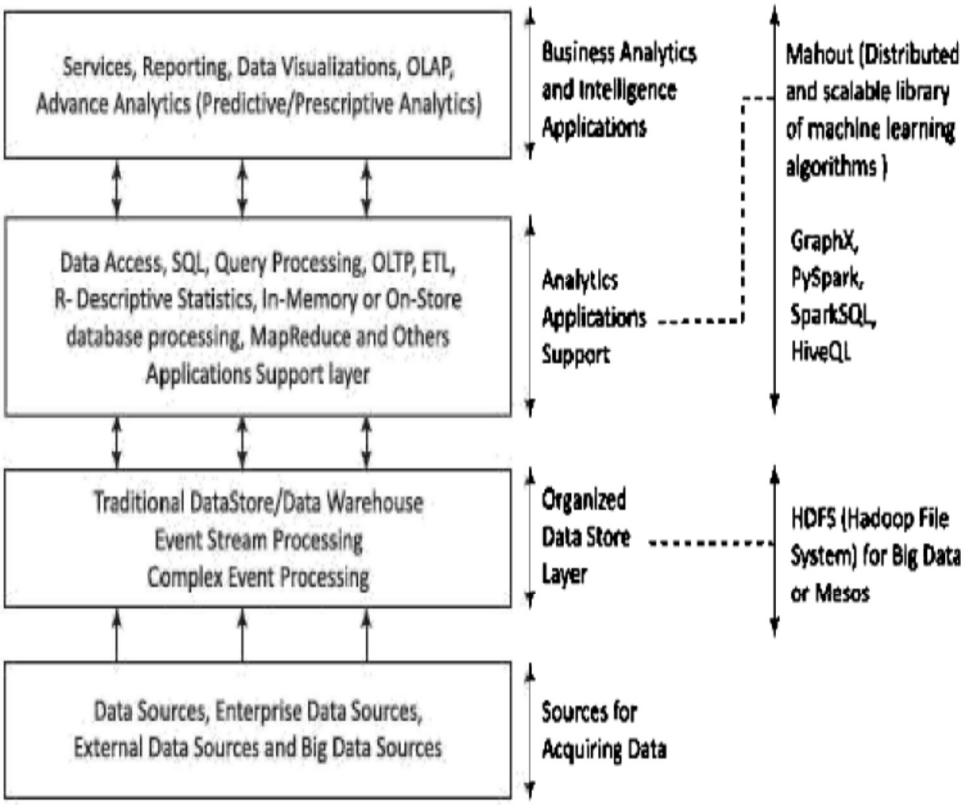



**Figure 1.9** Traditional and Big Data analytics architecture reference model

**Berkely Data Analysis Stack(BDAS)**

Berkeley Data Analytics Stack (BDAS) consists of data processing, data management and resource management layers. Following list these:

1. Applications, AMP-Genomics and Carat run at the BDAS. Data processing software component provides in-memory processing which processes the data efficiently across the frameworks. AMP stands for Berkeley's Algorithms, Machines and PeoplesLaboratory.
2. Data processing combines *batch, streaming* and *interactive* computations.
3. Resource management software component provides for sharing the infrastructure across various frameworks.

Figure 1.10 shows a four layers architecture for Big Data Stack that consists of Hadoop, MapReduce, Spark core and SparkSQL, Streaming, R, GraphX, MLib, Mahout, Arrow andKafka

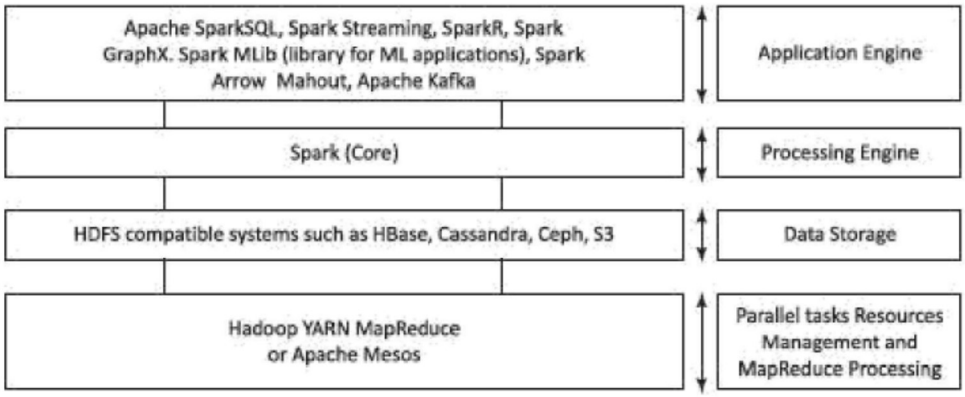



**Figure 1.10** Four layers architecture for Big Data Stack consisting of Hadoop, MapReduce, Spark core and SparkSQL, Streaming, R, GraphX, MLib, Mahout, Arrow and Kafka

## 1.7 Big Data Applications

### Big Data in Marketing and Sales

Data are important for most aspect of marketing, sales and advertising. Customer Value (CV) depends on three factors - quality, service and price. Big data analytics deploy large volume of data to identify and derive intelligence using predictive models about the individuals. The facts enable marketing companies to decide what products to sell.

A definition of marketing is the creation, communication and delivery of *value* to customers. Customer (desired) value means what a customer desires from a product. Customer (perceived) value means what the customer believes to have received from a product after purchase of the product. Customer value analytics (CVA) means analyzing what a customer really needs. CVA makes it possible for leading marketers, such as Amazon to deliver the consistent customer experiences.

### Big Data Analytics in Detection of Marketing Frauds

Big Data analytics enable fraud detection. Big Data usages has the following features-for enabling detection and prevention of frauds:

- Fusing of existing data at an enterprise data warehouse with data from sources such as social media, websites, biogs, e-mails, and thus enriching existing data
- Using multiple sources of data and connecting with many applications
- Providing greater insights using querying of the multiple source data
- Analyzing data which enable structured reports and visualization
- Providing high volume data mining, new innovative applications and thus leading to new business intelligence and knowledge discovery

### Big Data Risks

Large volume and velocity of Big Data provide greater insightsbut also associate risks with the data used. Data included may be erroneous, less accurate or far from reality. Analytics introduces new errors due to such data.

Five data risks, described by Bernard Marr are data security, data privacy breach, costs affecting

profits, bad analytics and bad data

**Big Data Credit Card Risk Management**

Financial institutions, such as banks, extend loans to industrial and household sectors. These institutions in many countries face credit risks, mainly risks of (i) loan defaults, (ii) timely return of interests and principal amount. Financing institutions are keen to get insights into the following:

1. Identifying high credit rating business groups and individuals,
2. Identifying risk involved before lending money
3. Identifying industrial sectors with greater risks
4. Identifying types of employees (such as daily wage earners in construction sites) and businesses (such as oil exploration) with greater risks
5. Anticipating liquidity issues (availability of money for further issue of credit and rescheduling credit installments) over the years.

**Big Data in Healthcare**

Big Data analytics in health care use the following data sources:clinical records, (ii) pharmacy records, (3) electronic medical records (4) diagnosis logs and notes and (v) additional data, such as deviations from person usual activities, medical leaves from job, social interactions. Healthcare analytics using Big Data can facilitate the following:

1. Provisioning of value-based and customer-centric healthcare,
2. Utilizing the 'Internet of Things' for health care
3. Preventing fraud, waste, abuse in the healthcare industry and reduce healthcare costs (Examples of frauds are excessive or duplicate claims for clinical and hospital treatments. Example of waste is unnecessary tests. Abuse means unnecessary use of medicines, such as tonics and testing facilities.)
4. Improving outcomes
5. Monitoring patients in real time.

**Big Data in Medicine**

Big Data analytics deploys large volume of data to identify and derive intelligence using predictive models about individuals. Big Data driven approaches help in research in medicine which can help patients

Following are some findings: building the health profiles of individual patients and predicting models for diagnosing better and offer better treatment,

Aggregating large volume and variety of information around from multiple sources the DNAs, proteins, and metabolites to cells, tissues, organs, organisms, and ecosystems, that can enhance the understanding of biology of diseases. Big data creates patterns and models by data mining and help in better understanding and research,

Deploying wearable devices data, the devices data records during active as well as inactive periods, provide better understanding of patient health, and better risk profiling the user for certain diseases.

**Big Data in Advertising**

The impact of Big Data is tremendous on the digital advertising industry. The digital advertising industry sends advertisements using SMS, e-mails, WhatsApp, Linkedln, Facebook, Twitter and other mediums.

Big Data captures data of multiple sources in large volume, velocity and variety of data unstructured and enriches the structured data at the enterprise data warehouse. Big data real time analytics provide emerging trends and patterns, and gain actionable insights for facing competitions from similar products. The data helps digital advertisers to discover new relationships, lesser competitive regions and areas.

Success from advertisements depend on collection, analyzing and mining. The new insights enable the personalization and targeting the online, social media and mobile for advertisements called hyper-localized advertising.

Advertising on digital medium needs optimization. Too much usage can also effect negatively. Phone calls, SMSs, e-mail-based advertisements can be nuisance if sent without appropriate researching on the potential targets. The analytics help in this direction. The usage of Big Data after appropriate filtering and elimination is crucial enabler of BigData Analytics with appropriate data, data forms and data handling in the right manner.

# Module -2

# Introduction to Hadoop

## 2.1 Big Data Programming Model

A programming model is centralized computing of data in which the data is transferred from multiple distributed data sources to a central server. Analyzing, reporting, visualizing, business-intelligence tasks compute centrally. Data are inputs to the central server.

Another programming model is distributed computing that uses the databases at multiple computing nodes with data sharing between the nodes during computation. Distributed computing in this model requires the cooperation (sharing) between the DBs in a transparent manner. Transparent means that each user within the system may access all the data within all databases as if they were a single database. A second requirement is location independence. Analysis results should be independent of geographical locations. The access of one computing node to other nodes may fail due to a single link failure.

**Distributed pieces of codes as well as the data at the computing nodes** Transparency between data nodes at computing nodes do not fulfil for Big Data when distributed computing takes place using data sharing between local and remote. Following are the reasons for this:

- Distributed data storage systems do not use the concept of joins.
- Data need to be fault-tolerant and data stores should take into account the possibilities of network failure. When data need to be partitioned into data blocks and written at one set of nodes, then those blocks need replication at multiple nodes. This takes care of possibilities of network faults. When a network fault occurs, then replicated node makes the data available.

Big Data follows a theorem known as the CAP theorem. The CAP states that out of three properties (consistency, availability and partitions), two must at least be present for applications, services and processes.

### i. Big Data Store Model

A model for Big Data store is as follows:

Data store in file system consisting of data blocks (physical division of data). The data blocks are distributed across multiple nodes. Data nodes are at the racks of a cluster. Racks are scalable.

A Rack has multiple data nodes (data servers), and each cluster is arranged in a number of racks.

Data Store model of files in data nodes in racks in the clusters Hadoop system uses the data store model in which storage is at clusters, racks, data nodes and data blocks. Data blocks replicate at the DataNodes such that a failure of link leads to access of the data block from the other nodes replicated at the same or other racks.

**ii. Big Data Programming Model**

Big Data programming model is that application in which application jobs and tasks (or sub-tasks) is scheduled on the same servers which store the data for processing.

## 2.2 Hadoop and its echo system

Hadoop is a computing environment in which input data stores, processes and stores the results. The environment consists of clusters which distribute at the cloud or set of servers. Each cluster consists of a string of data files constituting data blocks. The toy named Hadoop consisted of a stuffed elephant. The Hadoop system cluster stuffs files in data blocks. The complete system consists of a scalable distributed set of clusters.

Infrastructure consists of cloud for clusters. A cluster consists of sets of computers or PCs. The Hadoop platform provides a low cost Big Data platform, which is open source and uses cloud services. Tera Bytes of data processing takes just few minutes. Hadoop enables distributed processing of large datasets (above 10 million bytes) across clusters of computers using a programming model called MapReduce. The system characteristics are scalable, self-manageable, self-healing and distributed file system.

Scalable means can be scaled up (enhanced) by adding storage and processing units as per the requirements. Self-manageable means creation of storage and processing resources which are used, scheduled and reduced or increased with the help of the system itself. Self-healing means that in case of faults, they are taken care of by the system itself. Self-healing enables functioning and resources availability. Software detect and handle failures at the task level. Software enable the service or task execution even in case of communication or node failure.

### 2.1.1 Hadoop Core Components

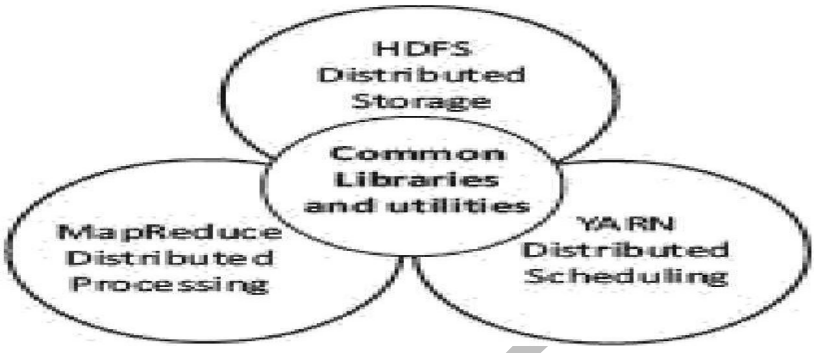



**Figure 2.1 Core components of Hadoop**

**The Hadoop core components of the framework are:**

**Hadoop Common -** The common module contains the libraries and utilities that are required by the other modules of Hadoop. For example, Hadoop common provides various components and interfaces for distributed file system and general input/output. This includes serialization, Java RPC (Remote Procedure Call) and file-based data structures.

**Hadoop Distributed File System (HDFS) -** A Java-based distributed file system which can store all kinds of data on the disks at the clusters.

**MapReduce vl -** Software programming model in Hadoop 1 using Mapper and Reducer. The vl processes large sets of data in parallel and in batches.

**YARN -** Software for managing resources for computing. The user application tasks or sub-tasks run in parallel at the Hadoop, uses scheduling and handles the requests for the resources in distributed running of the tasks.

**MapReduce v2 - Hadoop 2 YARN**-based system for parallel processing of large datasets and distributed processing of the application tasks.

### 2.2.2 Features of Hadoop

Hadoop features are as follows:

**1. Fault-efficient scalable, flexible and modular design** which uses simple and modular programming model. The system provides servers at high scalability. The system is scalable by adding new nodes to handle larger data. Hadoop proves very helpful in storing, managing,

processing and analyzing Big Data.

**2. Robust design of HDFS:** Execution of Big Data applications continue even when an individual server or cluster fails. This is because of Hadoop provisions for backup (due to replications at least three times for each data block) and a data recovery mechanism. HDFS thus has high reliability.

**3. Store and process Big Data:** Processes Big Data of 3V characteristics.

**4. Distributed clusters computing model with data locality:** Processes Big Data at high speed as the application tasks and sub-tasks submit to the DataNodes. One can achieve more computing power by increasing the number of computing nodes. The processing splits across multiple DataNodes (servers), and thus fast processing and aggregated results.

**5. Hardware fault-tolerant:** A fault does not affect data and application processing. If a node goes down, the other nodes take care of the residue. This is due to multiple copies of all data blocks which replicate automatically. Default is three copies of data blocks.

**6. Open-source framework:** Open source access and cloud services enable large data store. Hadoop uses a cluster of multiple inexpensive servers or the cloud.

**7. Java and Linux based:** Hadoop uses Java interfaces. Hadoop base is Linux but has its own set of shell commands support.

### 2.2.3. Hadoop Eco system Components

The four layers in Figure 2.2 are as follows:

(i) Distributed storage layer

(ii) Resource-manager layer for job or application sub-tasks scheduling and execution

(iii) Processing-framework layer, consisting of Mapper and Reducer for the MapReduce process-flow.

(iv) APIs at application support layer (applications such as Hive and Pig). The codes communicate and run using MapReduce or YARN at processing framework layer. Reducer output communicate to APIs (Figure 2.2).

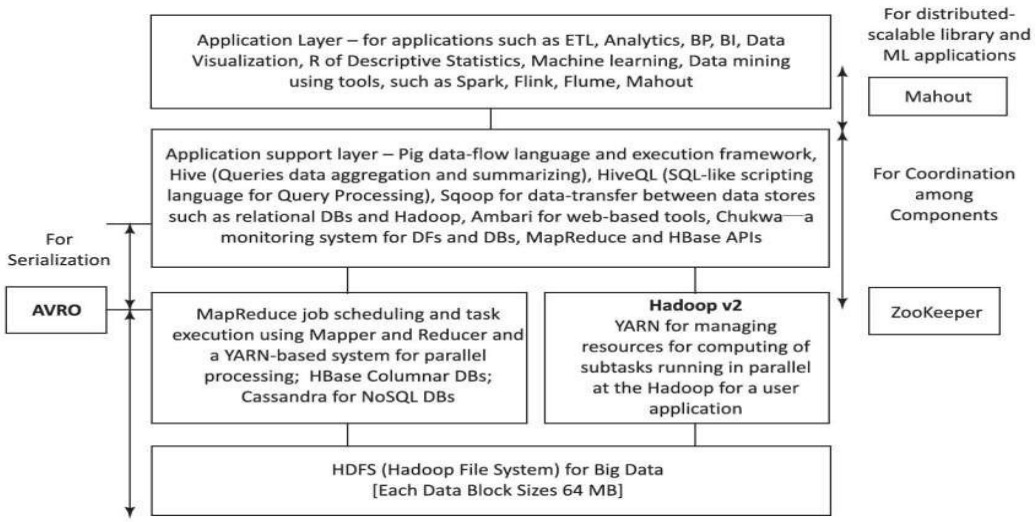



**Figure 2.2 Hadoop main components and ecosystem components**

AVRO enables data serialization between the layers. Zookeeper enables coordination among layer components.

The holistic view of Hadoop architecture provides an idea of implementation of Hadoop components of the ecosystem. Client hosts run applications using Hadoop ecosystem projects, such as Pig, Hive and Mahout.

## 2.3 HADOOP DISTRIBUTED FILE SYSTEM

HDFS is a core component of Hadoop. HDFS is designed to run on a cluster of computers and servers at cloud-based utility services.

HDFS stores Big Data which may range from GBs (1 GB= 230 B) to PBs (1 PB=

1015 B, nearly the 250 B). HDFS stores the data in a distributed manner in order to compute fast. The distributed data store in HDFS stores data in any format regardless of schema.

### 2.3.1 HDFS Storage

Hadoop data store concept implies storing the data at a number of dusters. Each cluster has a number of data stores, called racks. Each rack stores a number of DataNodes. Each DataNode has a large number of data blocks. The racks distribute across a cluster. The nodes have processing and storage capabilities. The nodes have the data in data blocks to run the application tasks. The data blocks replicate by default at least on three DataNodes in same or remote nodes.

Data at the stores enable running the distributed applications including analytics, data mining, OLAP using the clusters. A file, containing the data divides into data blocks. A data block default size is 64 MBs

**Hadoop HDFS features are as follows**

i. Create, append, delete, rename and attribute modification functions

ii. Content of individual file cannot be modified or replaced but appended with new data at the end of the file

iii. Write once but read many times during usages and processing

iv. Average file size can be more than 500 MB.

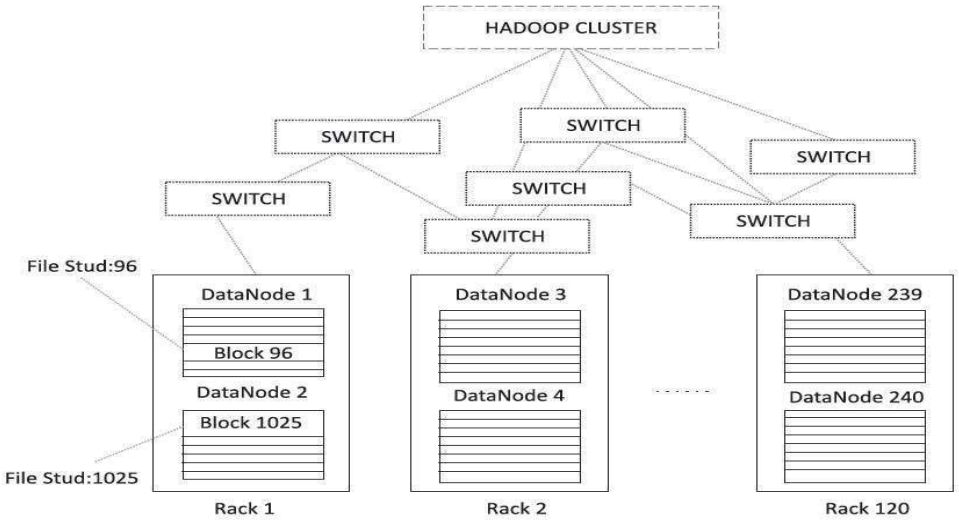



**Figure 2.3 A Hadoop cluster example,**

Consider a data storage for University students. Each student data, stuData which is in a file of size less than 64 MB (1 MB= 220 B). A data block stores the full file data for a student of stuData_idN, whereN = 1 to 500.

i. How the files of each student will be distributed at a Hadoop cluster? How many student data can be stored at one cluster? Assume that each rack has two DataNodes for processing each

of 64 GB (1 GB= $2^{30}$ B) memory. Assume that cluster consists of 120 racks, and thus 240 DataNodes.

ii. What is the total memory capacity of the cluster in TB ((1 TB= 240 B) and DataNodes in each rack?

iii. Show the distributed blocks for students with ID= 96 and 1025. Assume default replication in the DataNodes = 3.

iv. What shall be the changes when a stuData file sizes 128 MB?

**SOLUTION**

i. Data block default size is 64 MB. Each students file size is less than 64MB. Therefore, for each student file one data block suffices. A data block is in a DataNode. Assume, for simplicity, each rack has two nodes each of memory capacity = 64 GB. Each node can thus store 64 GB/64MB = 1024 data blocks = 1024 student files. Each rack can thus store 2 x 64 GB/64MB = 2048 data blocks = 2048 student files. Each data block default replicates three times in the DataNodes. Therefore, the number of students whose data can be stored in the cluster = number of racks multiplied by number of files divided by 3 = 120 x 2048/3 = 81920. Therefore, the maximum number of 81920 stuData_IDN files can be distributed per cluster, with N = 1 to 81920.

ii. Total memory capacity of the cluster = 120 x 128 MB = 15360 GB = 15 TB. Total memory capacity of each DataNode in each rack= 1024 x 64 MB= 64 GB.

iii. Figure 2.3 shows a Hadoop cluster example, and the replication of data blocks in racks for two students of IDs 96 and 1025. Each stuData file stores at two data blocks, of capacity 64 MB each.

iv. Changes will be that each node will have half the number of data blocks.

### 2.3.1.1 Hadoop Physical organization

Figure 2.4 shows the client, master NameNode, primary and secondary MasterNodes and slave nodes in the Hadoop physical architecture. Clients as the users run the application with the help of Hadoop ecosystem projects. For example, Hive, Mahout and Pig are the ecosystem's projects. They are not required to be present at the Hadoop cluster.

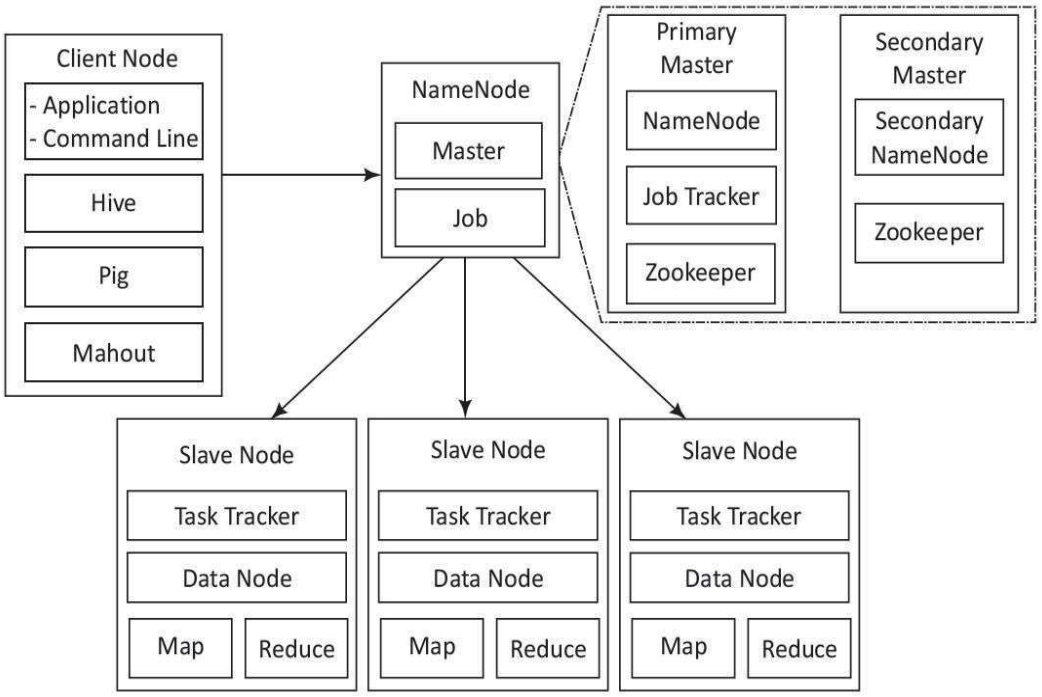



**Figure 2.4** The client, master NameNode, MasterNodes and slave nodes

A single MasterNode provides HDFS, MapReduce and Hbase using threads in small to medium sized clusters. When the cluster size is large, multiple servers are used, such as to balance the load. The secondary NameNode provides NameNode management services and Zookeeper is used by HBase for metadata storage.

The MasterNode fundamentally plays the role of a coordinator. The MasterNode receives client connections, maintains the description of the global file system namespace, and the allocation of file blocks. It also monitors the state of the system in order to detect any failure. The Masters consists of three components NameNode, Secondary NameNode and JobTracker. The NameNode stores all the file system related information such as:

- The file section is stored in which part of the cluster
- Last access time for the files
- User permissions like which user has access to the file.

Secondary NameNode is an alternate for NameNode. Secondary node keeps a copy of

NameNode meta data. Thus, stored meta data can be rebuilt easily, in case ofNameNode failure. TheJobTracker coordinates the parallel processing of data.

#### *2.3.1.1 Hadoop 2*

- Single Name Node failure in Hadoop 1 is an operational limitation.
- Scaling up was restricted to scale beyond a few thousands of DataNodes and number of Clusters.
- Hadoop 2 provides the multiple NameNodes which enables higher resources availability

### 2.3.1.2 HDFS commands

**Table 2.1** Examples of usages of commands

| HDFS shell command | Example of usage |
|---|---|
| -mkdir | Assume stu_filesdir is a directory of student files in Example 2.2. Then command for creating the directory is `$Hadoop hdfs-mkdir/user/stu_filesdir` creates the directory named stu_files_dir |
| -put | Assume file stuData_id96 to be copied at stu_filesdir directory in Example 2.2. Then `$Hadoop hdfs-put stuData_id96 /user/ stu_filesdir` copies file for student of id96 into stu_filesdir directory |
| -ls | Assume all files to be listed. Then `$hdfs hdfs dfs-ls` command does provide the listing. |
| -cp | Assume stuData_id96 to be copied from stu_filesdir to new students' directory newstu_filesDir. Then `$Hadoop hdfs-cp stuData_id96 /user/stu_filesdir newstu_filesDir` copies file for student of ID 96 into stu_filesdir directory |

## 2.4 MAPREDUCE FRAMEWORK AND PROGRAMMING MODEL

**Mapper means** software for doing the assigned task after organizing the data blocks imported using the keys. A key specifies in a command line of Mapper. The command maps the key to the data, which an application uses.

**Reducer means** software for reducing the mapped data by using the aggregation, query or user-specified function. The reducer provides a concise cohesive response for the application.

**Aggregation function means** the function that groups the values of multiple rows together to

result a single value of more significant meaning or measurement. For example, function such as count, sum, maximum, minimum, deviation and standard deviation.

**Querying function** means a function that finds the desired values. For example, function for finding a best student of a class who has shown the best performance in examination.

**MapReduce allows** writing applications to process reliably the huge amounts of data, in parallel, on large clusters of servers. The cluster size does not limit as such to process in parallel. The parallel programs of MapReduce are useful for performing large scale data analysis using multiple machines in the cluster.

Features o fMapReduce framework are as follows:

- Provides automatic parallelization and distribution of computation based on several processors
- Processes data stored on distributed clusters of DataNodes and racks
- Allows processing large amount of data in parallel
- Provides scalability for usages of large number of servers
- Provides Map Reduce batch-oriented programming model in Hadoop version 1
- Provides additional processing modes in Hadoop 2 YARN-based system and enables required parallel processing. For example, for queries, graph databases, streaming data, messages, real-time OLAP and ad hoc analytics with Big Data 3V characteristics.

## 2.5 HADOOP YARN

- YARN is a resource a management platform. It manages the computer resources.
- YARN manages the schedules for running the sub tasks. Each sub tasks uses the resources in the allotted interval time.
- YARN separates the resources management and processing components.
- It stands for YET ANOTHER RESOURCE NEGOTIATOR , it manages and allocates resources for the application sub tasks and submit the resources for them in the Hadoop system.

## Hadoop 2 Execution Model

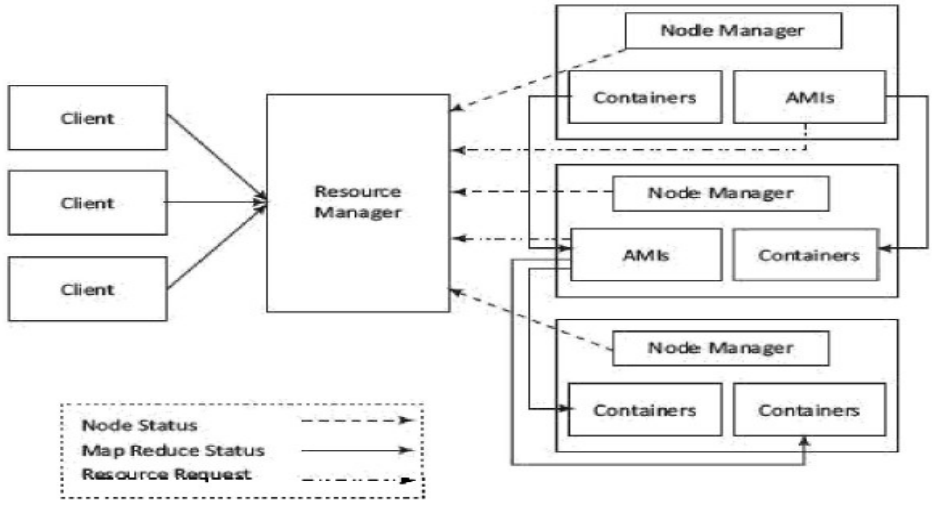



**Figure 2.5 YARN based Execution Model**

The figure shows the YARN components-Client, Resource Manager (RM), Node Manager (NM), Application Master (AM) and Containers.

Figure 2.5 also illustrates YARN components namely, Client, Resource Manager (RM), Node Manager (RM), Application Master (AM) and Containers.

List of actions of YARN resource allocation and scheduling functions is as follows:

A MasterNode has two components: (i) Job History Server and (ii) Resource Manager(RM).

A Client Node submits the request of an application to the RM. The RM is the master. One RM exists per cluster. The RM keeps information of all the slave NMs. Information is about the location (Rack Awareness) and the number of resources (data blocks and servers) they have. The RM also renders the Resource Scheduler service that decides how to assign the resources. It, therefore, performs resource management as well as scheduling.

Multiple NMs are at a cluster. An NM creates an AM instance (AMI) and starts up. The AMI initializes itself and registers with the RM. Multiple AMIs can be created in an AM.

The AMI performs role of an Application Manager (ApplM), that estimates the resources requirement for running an application program or sub- task. The ApplMs send their requests

for the necessary resources to the RM. Each NM includes several containers for uses by the subtasks of the application.

NM is a slave of the infrastructure. It signals whenever it initializes. All active NMs send the controlling signal periodically to the RM signaling their presence.

## 2.6 HADOOP ECOSYSTEM TOOLS

| | |
|---|---|
| ZooKeeper-Coordination service | Provisions high-performance coordination service for distributed running of applications and tasks |
| Avro-Data serialization and transfer utility | Provisions data serialization during data transfer between application and processing layers |
| Oozie | Provides a way to package and bundles multiple coordinator and workflow jobs and manage the lifecycle of those jobs |
| Sqoop (SQL-to-Hadoop)-A data-transfer software | Provisions for data-transfer between data stores such as relational DBs and Hadoop |
| Flume - Large data transfer utility | Provisions for reliable data transfer and provides for recovery in case of failure. Transfers large amount of data in applications, such as related to social-media messages |
| Ambari-A web-based tool | Provisions, monitors, manages, and viewing of functioning of the cluster, MapReduce, Hive and Pig APis |
| Chukwa-A data collection system | Provisions and manages data collection system for large and distributed systems |
| HBase-A structured data store using database | Provisions a scalable and structured database for large tables (Section 2.6.3) |
| Cassandra - A database | Provisions scalable and fault-tolerant database for multiple masters (Section 3.7) |

| | |
|---|---|
| Hive -A data warehouse system | Provisions data aggregation, data-summarization, data warehouse infrastructure, ad hoc (unstructured) querying and SQL-like scripting language for query processing using HiveQL (Sections 2.6.4, 4.4 and 4.5) |
| Pig-A high-level dataflow language | Provisions dataflow (DF) functionality and the execution framework for parallel computations |
| Mahout-A | Provisions scalable machine learning and library functions for data mining and analytics |

Module -1

chapter -1

# Hadoop Distributed file System Basics

Hadoop distributed file system was designed for Big data Processing. It is capable of supporting many users simultaneously. The design of HDFS is based on the design of the Google File System (GFS).

HDFS is designed for data streaming where large amount of data are read from the disk in bulk. The HDFS size is typically 64 mB or 128 mB.

## HDFS components

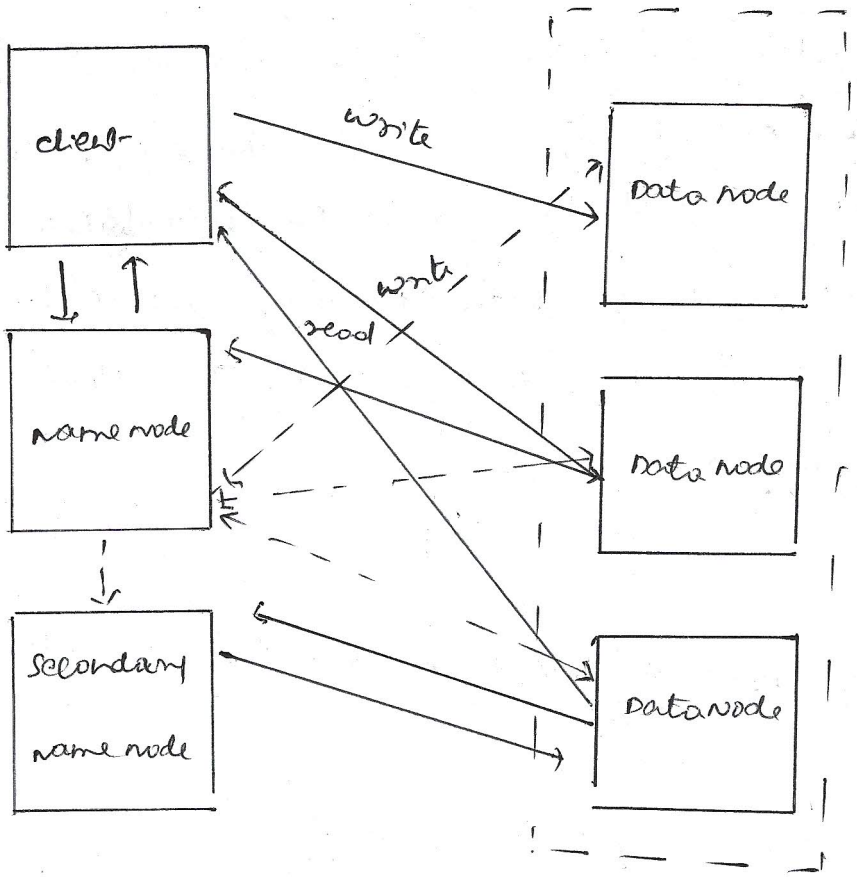
The design of HDFS is based on two types of nodes:

1) Name Node
2) Data Node

Name node manages all the metadata needed to store and retrieve actual data from the DataNode. No data is actually stored on the Name node.

master Name Node manages the file system namespace and regulate users to files by client. File system namespace operations such as opening, closing, and renaming files and directories are all managed by the namenode.

The slaves (Data Node) are responsible for serving read and write requests from the file system to the client. The name node manages block creation, deletion, and replication.




various system roles in an HDFS deployment.

Figure shows client / name node / Data node interaction.

When a client write data, it first communicates with the name node request to create a file. The name node determines how many blocks are needed and provides client with the Data Node that will store the data.

The name node will attempt to write replicas of the data blocks on nodes that are in other separate block. If there is only one rack, then the replicated blocks are written to other server in the same rack. After the Data node acknowledges that the file block replication is complete, the client closes the file and inform the name node that the operation is complete.

Reading data happens in a similar fashion. The client request a file from the Namenode, which returns the best DataNode from which to read the data. The client access the data directly from the DataNode.

once the meta data has been delivered to the client, the Namenode step back and lets the conversion b/w the client and the DataNode proceed. While the data transfer is progressing, the namenode also monitors the DataNode by listening for heart beats sent from DataNodes.

The purpose of the secondary name node is to perform periodic checkpoint that evaluate the status of the Name Node.

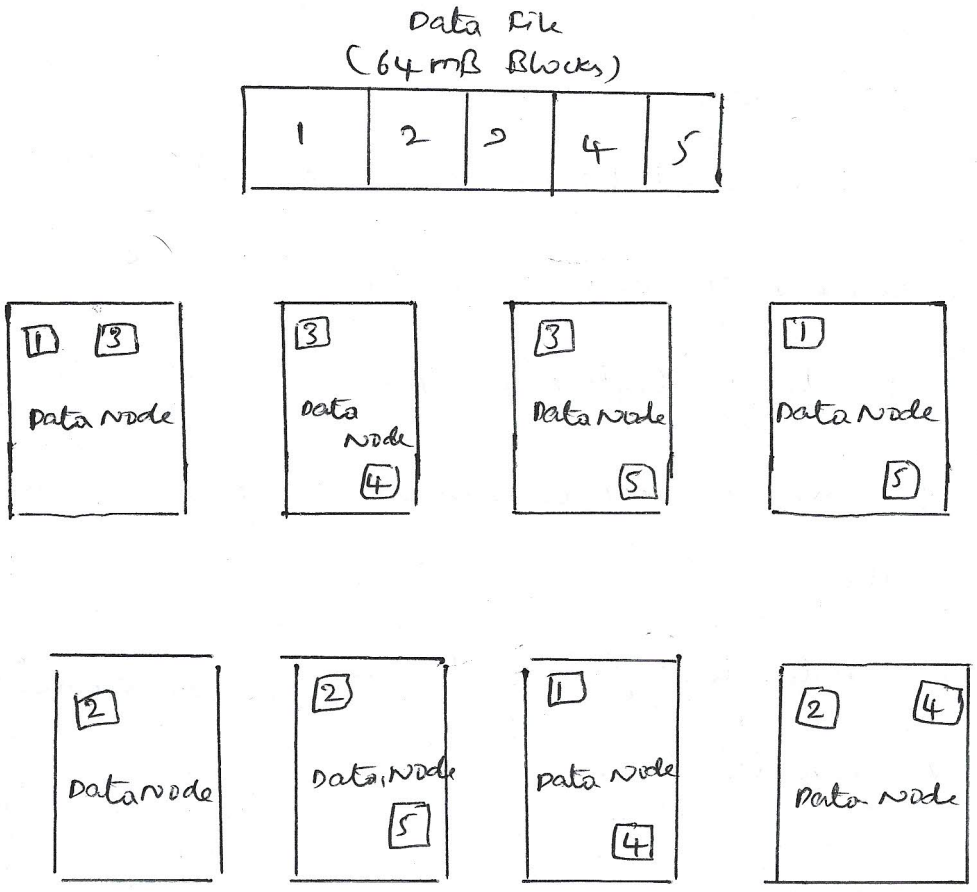
Various roles of HDFS

1. HDFS uses a master/slave model designed for large file reading/streaming
2. The Name Node is a metadata servers.
3. HDFS provides a single namespace that is managed by the Namenode
4. Hdfs provides a single namespace that is managed by the Namenode.
5. Data is redundantly stored on DataNode. There is no data on the Name Node
6. Secondary Name Node performs checkpoints of the Name Node file System's

# HDFS Block Replication

When HDFS writes a file, it is replicated across the cluster. Hadoop clusters containing more than eight Data Nodes, the replication value is usually set to 3. Hadoop cluster of eight or fewer DataNode but more than one Data Node, a replication factor is 2. For a single machine replication factor is 1.

HDFS default block size is 64 mB. In typical OS the block size is 4 KB or 8 KB. The HDFS default block size is not the minimum block size. If a 20 KB file is written to HDFS it will create a block that is approximately 20 KB only. If a file size is 80 mB, a 64 mB block and a 16 mB block will be created.




HDFS block replication example

Figure provides an example of how a file is broken into blocks and replicated across the cluster. In this case, a replication factor of 3 ensures that any one data node can fail and the replicated blocks will be available on the other nodes — and then subsequent re-replicated on other data node.

## HDFS Safe mode

When the name node starts, it enter a read-only safe mode where blocks can't be replicated or deleted. Safe mode enables the NameNode to perform two important process.

1. The previous file system state is reconstructed by loading the fs image file into memory and replying the edit log.
2. The mapping between blocks and data nodes is created by waiting for enough of the DataNode's to register so that at least one copy of the data is available. Not all the DataNodes are required to register before HDFS exists from safe mode. The registration process may continue for some time.

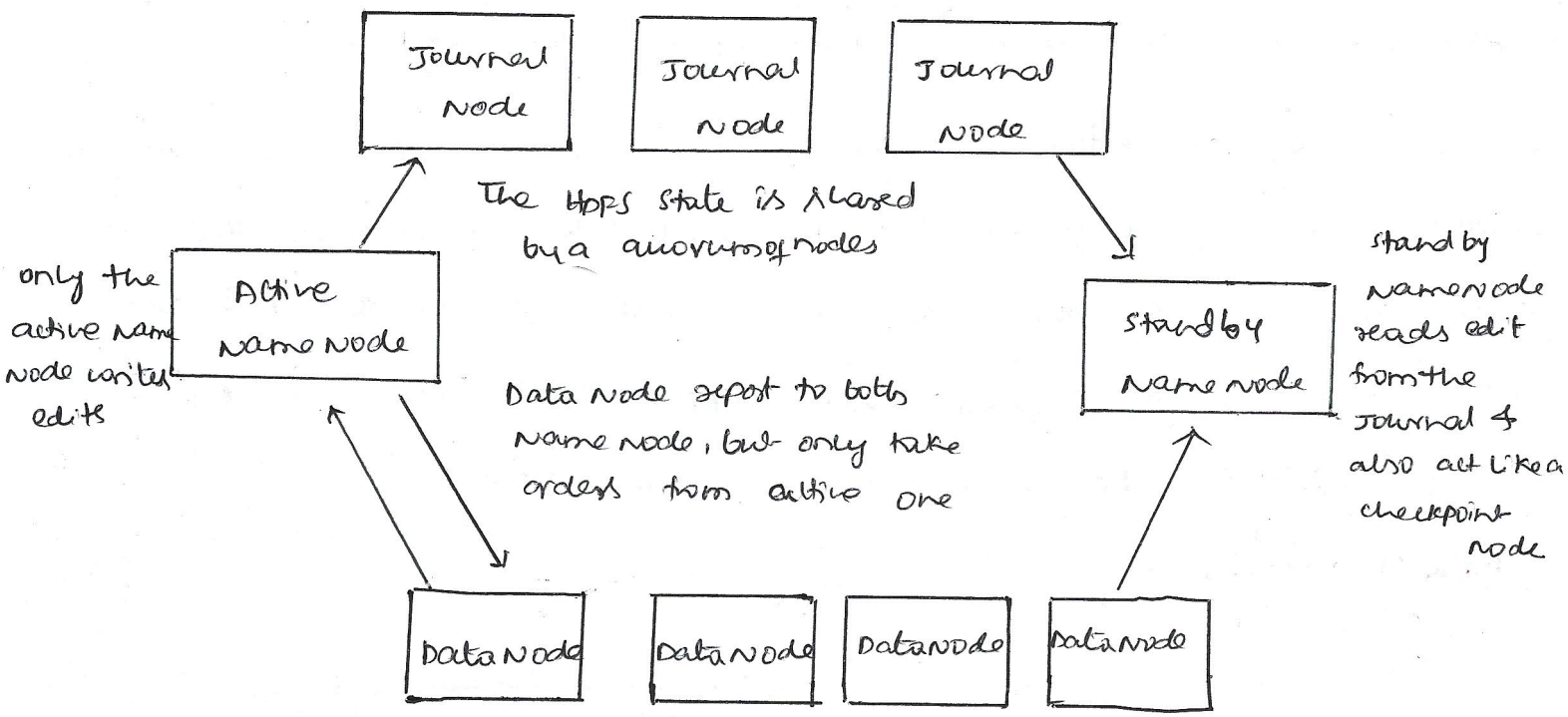
## Rack Awareness

Rack awarenes deals with data locality. Recall that one one of the main design goals of Hadoop MapReduce is to move the computation to the data. Assuming that most data center network don't ~~[illegible]~~ offer full bisection bandwidth, a typical Hadoop cluster will exhibit three levels of data locality.

1. Data resides on the local machine (best)
2. Data resides on the same rack (better)
3. Data resides in a different rack (good).

## Name node High Availability

The name node was a single point of failure that could bring down the entire Hadoop cluster. Name node hardware often employed redundant power supplies and storage to guard against such problems, but it was still susceptible to other failure. The solution was to implement Name node High Availability (HA) as a means to provide true failover service.




HDFS High Availability design

An HA Hadoop cluster has two (or more) separate Name node machines. Each machine is configured with exactly the same software

One of the Name Node machines is in the Active state, and other is in the Standby state. Like a single Name Node cluster, the Active Name node is responsible for all client HDFS operations in the cluster. The Standby Name node maintains enough state to provide a fast failover.

To guarante the file system state is preserved, bothe the Active and Standby Name node receive block reports from the Data Node. The Active node also sends all file system edit to a quorum of Jour -nal nodes.

To prevent confusion between Name node, the Journal nodes allow only one Name node to be a writer at a time. During ~~failover~~ failover, the Name node that is chosen to become active takes over the role of writing to the Journal nodes. A secondary Name node is not required in the HA configuration because the Standby node also perform the task of the secondary Name node.

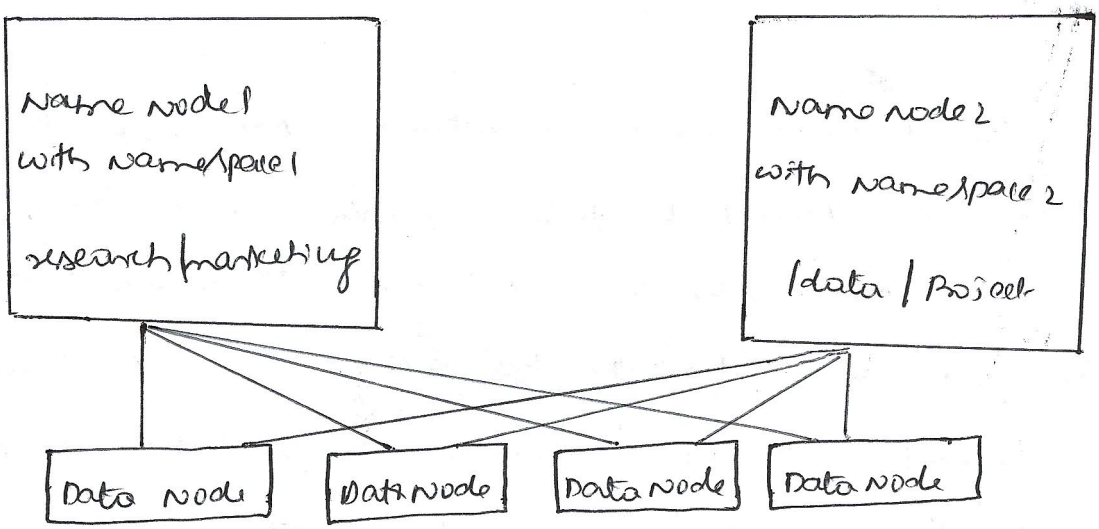
## HDFS Name node Federation

Another important feature of HDFS is Name node Federation. HDFS provided a single name space for the entire cluster managed by a single Name node. Thus resource of a single Name node determine the size of the name space. The key benefits are as follows

- Name space scalability :- HDFS cluster storage scales horizontally without placing a burden on the Name node.

- Better performance :- Adding more name node to the cluster scales the file system read/write operations throughput by separating the total name space.

- System isolation :- multiple name nodes enable different categories of applications to be distinguished, and user can be isolated to different name space.




HDFS Name node federation

Figure illustrates how HDFS Name node federation is accomplished. Name node1 manages the /research and /marketing name spaces and Name node2 manages the /data & /Project name space. The Name node don't communicate with each other and the Data node1 " just store data blocks" as directed by either Name node.

## HDFS checkpoints and Backup

Name node stores the metadata of the HDFS file system in a file called fsimage. File system modifications are written to an edit log file and at startup the Name Node merges the edits into a new fsimage. The secondary Name Node or checkpoint-Node periodically fetches edits from the Name Node, merge them and returns an updated fsimage to the Name node.

An HDFS Backup Node is similar, but also maintains an up-to-date copy of the file system name space both in memory and on disk. A Name node support one Backup node at a time. No checkpoint nodes may be registered if a Backup node is in use.

## HDFS Snapshots

HDFS snapshots are similar to backups, but are created by administrators using the hdfs dfs -snapshot command. HDFS snapshots are read-only point-in-time copies of the file system. They offer following features.

1. Snapshots can be taken of a sub-tree of the file system or the entire file system.
2. Snapshots can be used for data backup, protection against user errors, and disaster recovery.
3. Snapshots creation is instantaneous.

4. Blocks on the data node are not copied, because the snapshot files recorded the block list and the file size.

5. Snap shots donot adversely affect regular HDFS operations.

## HDFS NFS Gateway

The HDFS NFS Gateway supports NFSV3 and enables HDFS to be mounted as part of the client's local file system. Users can browse the HDFS file system through their local file system that provide an NFSV3 client compatible OS. This feature offers users the following capabilities:

1. User can easily download / upload files from / to the HDFS file system to/from their local file system.

2. User can stream data directly to HDFS through the mount point. Appending to a file is supported, but random write capability is not supported.

## Module 2
# 1. Essential Hadoop Tools

**In This Chapter:**

- The Pig scripting tool is introduced as a way to quickly examine data both locally and on a Hadoop cluster.
- The Hive SQL-like query tool is explained using two examples.
- The Sqoop RDBMS tool is used to import and export data from MySQL to/from HDFS.
- The Flume streaming data transport utility is configured to capture weblog data into HDFS.
- The Oozie workflow manager is used to run basic and complex Hadoop workflows.
- The distributed HBase database is used to store and access data on a Hadoop cluster.

## USING APACHE PIG

Apache Pig is a high-level language that enables programmers to write complex MapReduce transformations using a simple scripting language. Pig Latin (the actual language) defines a set of transformations on a data set such as aggregate, join, and sort.

Apache Pig has several usage modes.

- The first is a local mode in which all processing is done on the local machine.
- The non-local (cluster) modes are MapReduce and Tez. These modes execute the job on the cluster using either the MapReduce engine or the optimized Tez engine.

There are also interactive and batch modes available; they enable Pig applications to be developed locally in interactive modes, using small amounts of data, and then run at scale on the cluster in a production mode. The modes are summarized in Table 7.1.

| | Local Mode | Tez Local Mode | MapReduce Mode | Tez Mode |
|---|---|---|---|---|
| Interactive Mode | Yes | Experimental | Yes | Yes |
| Batch Mode | Yes | Experimental | Yes | Yes |

Table 7.1 **Apache Pig Usage Modes**

### Pig Example Walk-Through

In this simple example, Pig is used  The following example assumes the user is hdfs, but any valid user with access to HDFS can run the example.

To begin the example, copy the passwd file to a working directory for local Pig operation:

```
$ cp /etc/passwd .
```

Next, copy the data file into HDFS for Hadoop MapReduce operation:

```
$ hdfs dfs -put passwd passwd
```

You can confirm the file is in HDFS by entering the following command:

```
hdfs dfs -ls passwd
-rw-r--r--   2 hdfs hdfs       2526 2015-03-17 11:08 passwd
```

In the following example of local Pig operation, all processing is done on the local machine (Hadoop is not used). First, the interactive command line is started:

```
$ pig -x local
```

If Pig starts correctly, you will see a grunt> prompt. Next, enter the following commands to load the passwd file and then grab the user name and dump it to the terminal. Note that Pig commands must end with a semicolon (;).

```
grunt> A = load 'passwd' using PigStorage(':');
grunt> B = foreach A generate $0 as id;
grunt> dump B;
```

The processing will start and a list of user names will be printed to the screen. To exit the interactive session, enter the command quit.

```
$ grunt> quit
```

To use Hadoop MapReduce, start Pig as follows (or just enter pig):

```
$ pig -x mapreduce
```

The same sequence of commands can be entered at the grunt> prompt. You may wish to change the $0 argument to pull out other items in the passwd file. Also, because we are running this application under Hadoop, make sure the file is placed in HDFS.

If you are using the Hortonworks HDP distribution with tez installed, the tez engine can be used as follows:

```
$ pig -x tez
```

Pig can also be run from a script. This script, which is repeated here, is designed to do the same things as the interactive version:

```
/* id.pig */
A = load 'passwd' using PigStorage(':');  -- load the passwd file
B = foreach A generate $0 as id;  -- extract the user IDs
dump B;
store B into 'id.out'; -- write the results to a directory name id.out
```

Comments are delineated by /* */ and -- at the end of a line. First, ensure that the id.out directory is not in your local directory, and then start Pig with the script on the command line:

```
$ /bin/rm -r id.out/
$ pig -x local id.pig
```

If the script worked correctly, you should see at least one data file with the results and a zero-length file with the name _SUCCESS. To run the MapReduce version, use the same procedure; the only difference is that now all reading and writing takes place in HDFS.

```
$ hdfs dfs -rm -r id.out
$ pig id.pig
```

## USING APACHE HIVE

Apache Hive is a data warehouse infrastructure built on top of Hadoop for providing data summarization, ad hoc queries, and the analysis of large data sets using a SQL-like language called HiveQL. Hive offers the following features:

- Tools to enable easy data extraction, transformation, and loading (ETL)
- A mechanism to impose structure on a variety of data formats

■ Access to files stored either directly in HDFS or in other data storage systems such as HBase

■ Query execution via MapReduce and Tez (optimized MapReduce)

**Hive Example Walk-Through**

To start Hive, simply enter the hive command. If Hive starts correctly, you should get a hive> prompt.

```
$ hive
(some messages may show up here)
hive>
```

As a simple test, create and drop a table. Note that Hive commands must end with a semicolon (;).

```
hive> CREATE TABLE pokes (foo INT, bar STRING);
OK
Time taken: 1.705 seconds
hive> SHOW TABLES;
OK
pokes
Time taken: 0.174 seconds, Fetched: 1 row(s)
hive> DROP TABLE pokes;
OK
Time taken: 4.038 seconds
```

A more detailed example can be developed using a web server log file to summarize message types. First, create a table using the following command:

```
hive> CREATE TABLE logs(t1 string, t2 string, t3 string, t4 string, t5 string, t6 string, t7 string) ROW
FORMAT DELIMITED FIELDS TERMINATED BY ' ';
OK
Time taken: 0.129 seconds
```

Next, load the data—in this case, from the sample.log file. Note that the file is found in the local directory and not in HDFS.

```
hive> LOAD DATA LOCAL INPATH 'sample.log' OVERWRITE INTO TABLE logs;
Loading data to table default.logs
Table default.logs stats: [numFiles=1, numRows=0, totalSize=99271, rawDataSize=0]
OK
Time taken: 0.953 seconds
```

Finally, apply the select step to the file. Note that this invokes a Hadoop MapReduce operation. The results appear at the end of the output (e.g., totals for the message types DEBUG, ERROR, and so on).

```
hive> SELECT t4 AS sev, COUNT(*) AS cnt FROM logs WHERE t4 LIKE '[%' GROUP BY t4;
Query ID = hdfs_20150327130000_d1e1a265-a5d7-4ed8-b785-2c6569791368
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
```

```
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1427397392757_0001, Tracking URL = http://norbert:8088/proxy/
application_1427397392757_0001/
Kill Command = /opt/hadoop-2.6.0/bin/hadoop job  -kill job_1427397392757_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2015-03-27 13:00:17,399 Stage-1 map = 0%,  reduce = 0%
2015-03-27 13:00:26,100 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.14 sec
2015-03-27 13:00:34,979 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.07 sec
MapReduce Total cumulative CPU time: 4 seconds 70 msec
Ended Job = job_1427397392757_0001
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.07 sec   HDFS Read: 106384
HDFS Write: 63 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 70 msec
OK
[DEBUG] 434
[ERROR] 3
[FATAL] 1
[INFO]  96
[TRACE] 816
[WARN]  4
Time taken: 32.624 seconds, Fetched: 6 row(s)
```

To exit Hive, simply type exit;

```
hive> exit;
```

### A More Advanced Hive Example

In this example, 100,000 records will be transformed from userid, movieid, rating, unixtime to userid, movieid, rating, and weekday using Apache Hive and a Python program (i.e., the UNIX time notation will be transformed to the day of the week). The first step is to download and extract the data:

```
$ wget http://files.grouplens.org/datasets/movielens/ml-100k.zip
$ unzip ml-100k.zip
$ cd ml-100k
```

Before we use Hive, we will create a short Python program called weekday_mapper.py with following contents:

```
import sys
import datetime

for line in sys.stdin:
  line = line.strip()
  userid, movieid, rating, unixtime = line.split('\t')
  weekday = datetime.datetime.fromtimestamp(float(unixtime)).isoweekday()
  print '\t'.join([userid, movieid, rating, str(weekday)])LOAD DATA LOCAL INPATH './u.data'
OVERWRITE INTO TABLE u_data;
```

Next, start Hive and create the data table (u_data) by entering the following at the hive> prompt:

```
CREATE TABLE u_data (
  userid INT,
  movieid INT,
```

```
  rating INT,
  unixtime STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE;
```

Load the movie data into the table with the following command:

```
hive> LOAD DATA LOCAL INPATH './u.data' OVERWRITE INTO TABLE u_data;
```

The number of rows in the table can be reported by entering the following command:

```
hive > SELECT COUNT(*) FROM u_data;
```

This command will start a single MapReduce job and should finish with the following lines:

```
...
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 2.26 sec   HDFS Read: 1979380
HDFS Write: 7 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 260 msec
OK
100000
Time taken: 28.366 seconds, Fetched: 1 row(s)
```

Now that the table data are loaded, use the following command to make the new table (u_data_new):

```
hive> CREATE TABLE u_data_new (
  userid INT,
  movieid INT,
  rating INT,
  weekday INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t';
```

The next command adds the weekday_mapper.py to Hive resources:

```
hive> add FILE weekday_mapper.py;
```

Once weekday_mapper.py is successfully loaded, we can enter the transformation query:

```
hive> INSERT OVERWRITE TABLE u_data_new
SELECT
  TRANSFORM (userid, movieid, rating, unixtime)
  USING 'python weekday_mapper.py'
  AS (userid, movieid, rating, weekday)
FROM u_data;
```

If the transformation was successful, the following final portion of the output should be displayed:

```
...
Table default.u_data_new stats: [numFiles=1, numRows=100000, totalSize=1179173,
rawDataSize=1079173]
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 3.44 sec   HDFS Read: 1979380 HDFS Write:
1179256 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 440 msec
OK
Time taken: 24.06 seconds
```

The final query will sort and group the reviews by weekday:

```
hive> SELECT weekday, COUNT(*) FROM u_data_new GROUP BY weekday;
```

Final output for the review counts by weekday should look like the following:

```
...
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 2.39 sec   HDFS Read: 1179386
HDFS Write: 56 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 390 msec
OK
1       13278
2       14816
3       15426
4       13774
5       17964
6       12318
7       12424
Time taken: 22.645 seconds, Fetched: 7 row(s)
```

As shown previously, you can remove the tables used in this example with the DROP TABLE command. In this case, we are also using the -e command-line option. Note that queries can be loaded from files using the -f option as well.

```
$ hive -e 'drop table u_data_new'
$ hive -e 'drop table u_data'
```

## USING APACHE SQOOP TO ACQUIRE RELATIONAL DATA

Sqoop is a tool designed to transfer data between Hadoop and relational databases. You can use Sqoop to import data from a relational database management system (RDBMS) into the Hadoop Distributed File System (HDFS), transform the data in Hadoop, and then export the data back into an RDBMS.

Sqoop can be used with any Java Database Connectivity (JDBC)–compliant database and has been tested on Microsoft SQL Server, PostgresSQL, MySQL, and Oracle.

### Apache Sqoop Import and Export Methods

Figure 7.1 describes the Sqoop data import (to HDFS) process. The data import is done in two steps. In the first step, shown in the figure, Sqoop examines the database to gather the necessary metadata for the data to be imported. The second step is a map-only (no reduce step) Hadoop job that Sqoop submits to the cluster. This job does the actual data transfer using the metadata captured in the previous step. Note that each node doing the import must have access to the database.

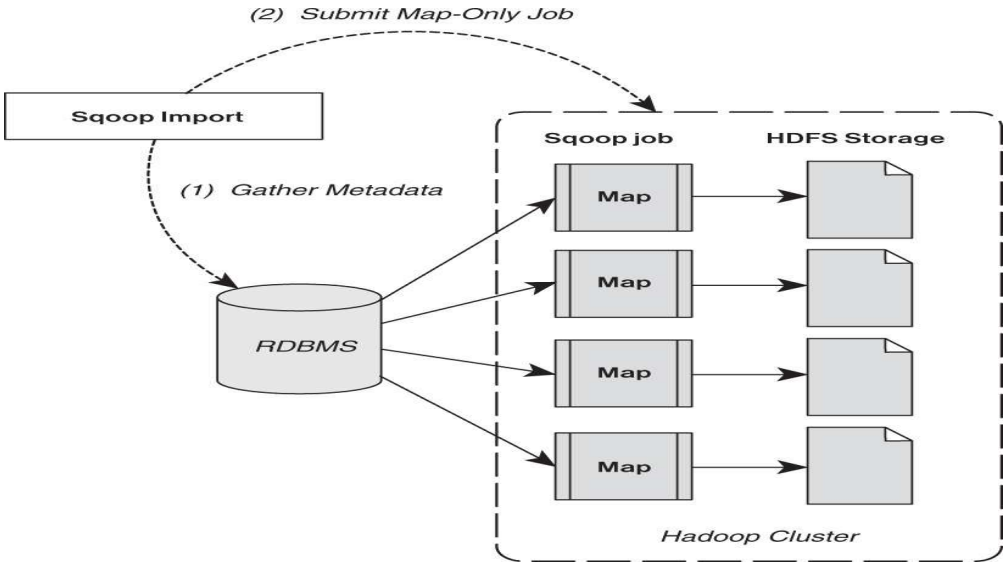



Figure 7.1 Two-step Apache Sqoop data import method (Adapted from Apache Sqoop Documentation)

The imported data are saved in an HDFS directory. Sqoop will use the database name for the directory, or the user can specify any alternative directory where the files should be populated. By default, these files contain comma-delimited fields, with new lines separating different records. You can easily override the format in which data are copied over by explicitly specifying the field separator and record terminator characters. Once placed in HDFS, the data are ready for processing.

Data export from the cluster works in a similar fashion. The export is done in two steps, as shown in Figure 7.2. As in the import process, the first step is to examine the database for metadata. The export step again uses a map-only Hadoop job to write the data to the database. Sqoop divides the input data set into splits, then uses individual map tasks to push the splits to the database. Again, this process assumes the map tasks have access to the database.

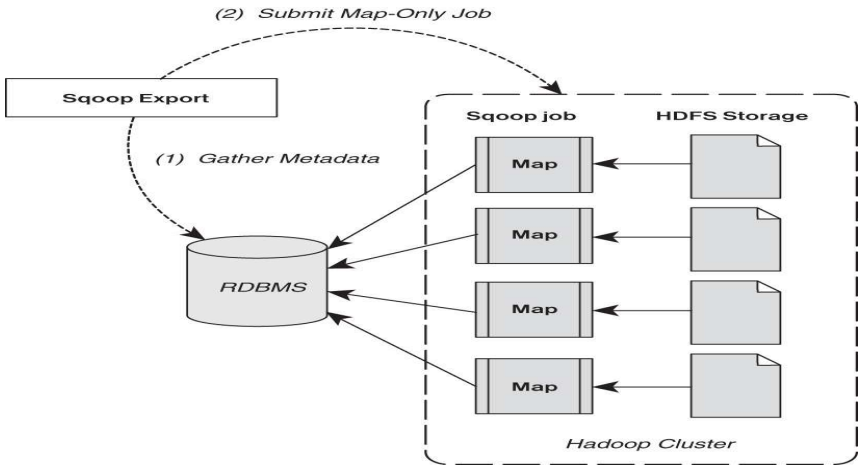


Figure 7.2 Two-step Sqoop data export method (Adapted from Apache Sqoop Documentation)

### Apache Sqoop Version Changes

Sqoop Version 1 uses specialized connectors to access external systems. These connectors are often optimized for various RDBMSs or for systems that do not support JDBC. Connectors are plug-in components based on Sqoop's extension framework and can be added to any existing Sqoop installation. Once a connector is installed, Sqoop can use it to efficiently transfer data between Hadoop and the external store supported by the connector. By default, Sqoop version 1 includes connectors for popular databases such as MySQL, PostgreSQL, Oracle, SQL Server, and DB2. It also supports direct transfer to and from the RDBMS to HBase or Hive.

In contrast, to streamline the Sqoop input methods, Sqoop version 2 no longer supports specialized connectors or direct import into HBase or Hive. All imports and exports are done through the JDBC interface. Table 7.2 summarizes the changes from version 1 to version 2. Due to these changes, any new development should be done with Sqoop version 2.

| Feature | Sqoop Version 1 | Sqoop Version 2 |
|---|---|---|
| Connectors for all major RDBMSs | Supported. | Not supported. Use the generic JDBC connector. |
| Kerberos security integration | Supported. | Not supported. |
| Data transfer from RDBMS to Hive or HBase | Supported. | Not supported. First import data from RDBMS into HDFS, then load data into Hive or HBase manually. |
| Data transfer from Hive or HBase to RDBMS | Not supported. First export data from Hive or HBase into HDFS, and then use Sqoop for export. | Not supported. First export data from Hive or HBase into HDFS, then use Sqoop for export. |

Table 7.2 **Apache Sqoop Version Comparison**

### Sqoop Example Walk-Through

The following simple example illustrates use of Sqoop

#### Step 1: Load Sample MySQL Database

```
$ wget http://downloads.mysql.com/docs/world_innodb.sql.gz
$ gunzip world_innodb.sql.gz
```

Next, log into MySQL (assumes you have privileges to create a database) and import the desired database by following these steps:

```
$ mysql -u root -p
mysql> CREATE DATABASE world;
mysql> USE world;
mysql> SOURCE world_innodb.sql;
mysql> SHOW TABLES;
+-----------------+
| Tables_in_world |
+-----------------+
| City            |
| Country         |
| CountryLanguage |
+-----------------+
3 rows in set (0.01 sec)
```

The following MySQL command will let you see the table details.

### Step 2: Add Sqoop User Permissions for the Local Machine and Cluster

In MySQL, add the following privileges for user sqoop to MySQL. Note that you must use both the local host name and the cluster subnet for Sqoop to work properly. Also, for the purposes of this example, the sqoop password is sqoop.

```
mysql> GRANT ALL PRIVILEGES ON world.* To 'sqoop'@'limulus' IDENTIFIED BY 'sqoop';
mysql> GRANT ALL PRIVILEGES ON world.* To 'sqoop'@'10.0.0.%' IDENTIFIED BY 'sqoop';
mysql> quit
```

Next, log in as sqoop to test the permissions:

```
$ mysql -u sqoop -p
  mysql> USE world;
  mysql> SHOW TABLES;
  +-----------------+
  | Tables_in_world |
  +-----------------+
  | City            |
  | Country         |
  | CountryLanguage |
  +-----------------+
  3 rows in set (0.01 sec)

  mysql> quit
```

### Step 3: Import Data Using Sqoop

As a test, we can use Sqoop to list databases in MySQL. The results appear after the warnings at the end of the output. Note the use of local host name (limulus) in the JDBC statement.

```
$ sqoop list-databases --connect jdbc:mysql://limulus/world --username sqoop --password sqoop
  Warning: /usr/lib/sqoop/../accumulo does not exist! Accumulo imports will fail.
  Please set $ACCUMULO_HOME to the root of your Accumulo installation.
  14/08/18 14:38:55 INFO sqoop.Sqoop: Running Sqoop version: 1.4.4.2.1.2.1-471
  14/08/18 14:38:55 WARN tool.BaseSqoopTool: Setting your password on the
command-line is insecure. Consider using -P instead.
  14/08/18 14:38:55 INFO manager.MySQLManager: Preparing to use a MySQL streaming
resultset.
  information_schema
```

```
test
world
```

In a similar fashion, you can use Sqoop to connect to MySQL and list the tables in the world database:

```
sqoop list-tables --connect jdbc:mysql://limulus/world --username sqoop --password sqoop
...
14/08/18 14:39:43 INFO sqoop.Sqoop: Running Sqoop version: 1.4.4.2.1.2.1-471
14/08/18 14:39:43 WARN tool.BaseSqoopTool: Setting your password on the
command-line is insecure. Consider using -P instead.
14/08/18 14:39:43 INFO manager.MySQLManager: Preparing to use a MySQL streaming
resultset.
City
Country
CountryLanguage
```

To import data, we need to make a directory in HDFS:

```
$ hdfs dfs -mkdir sqoop-mysql-import
```

The following command imports the Country table into HDFS. The option -table signifies the table to import, --target-dir is the directory created previously, and -m 1 tells Sqoop to use one map task to import the data.

```
$ sqoop import --connect jdbc:mysql://limulus/world  --username sqoop --password sqoop --table
Country  -m 1 --target-dir /user/hdfs/sqoop-mysql-import/country
...
14/08/18 16:47:15 INFO mapreduce.ImportJobBase: Transferred 30.752 KB in
12.7348 seconds
(2.4148 KB/sec)
14/08/18 16:47:15 INFO mapreduce.ImportJobBase: Retrieved 239 records.
```

The import can be confirmed by examining HDFS:

```
$ hdfs dfs -ls sqoop-mysql-import/country
Found 2 items
-rw-r--r--   2 hdfs hdfs          0 2014-08-18 16:47 sqoop-mysql-import/
world/_SUCCESS
-rw-r--r--   2 hdfs hdfs      31490 2014-08-18 16:47 sqoop-mysql-import/world/
part-m-00000
```

The file can be viewed using the hdfs dfs -cat command:

```
$ hdfs dfs -cat sqoop-mysql-import/country/part-m-00000
ABW,Aruba,North America,Caribbean,193.0,null,103000,78.4,828.0,793.0,Aruba,
Nonmetropolitan
Territory of The Netherlands,Beatrix,129,AW
...
ZWE,Zimbabwe,Africa,Eastern Africa,390757.0,1980,11669000,37.8,5951.0,8670.0,
Zimbabwe,
Republic,Robert G. Mugabe,4068,ZW
```

To make the Sqoop command more convenient, you can create an options file and use it on the command line. Such a file enables you to avoid having to rewrite the same options. For

example, a file called world-options.txt with the following contents will include the import command, --connect, --username, and --password options:

```
import
--connect
jdbc:mysql://limulus/world
--username
sqoop
--password
sqoop
```

The same import command can be performed with the following shorter line:

```
$ sqoop  --options-file world-options.txt --table City  -m 1 --target-dir /user/hdfs/sqoop-mysql-import/city
```

It is also possible to include an SQL Query in the import step. For example, suppose we want just cities in Canada:

```
SELECT ID,Name from City WHERE CountryCode='CAN'
```

In such a case, we can include the --query option in the Sqoop import request. The --query option also needs a variable called $CONDITIONS, which will be explained next. In the following query example, a single mapper task is designated with the -m 1 option:

```
sqoop  --options-file world-options.txt -m 1 --target-dir /user/hdfs/sqoop-mysql-import/canada-city --
query "SELECT ID,Name from City WHERE CountryCode='CAN' AND \$CONDITIONS"
```

Inspecting the results confirms that only cities from Canada have been imported:

```
$ hdfs dfs -cat sqoop-mysql-import/canada-city/part-m-00000
1810,MontrÄal
1811,Calgary
1812,Toronto
...
1856,Sudbury
1857,Kelowna
1858,Barrie
```

Since there was only one mapper process, only one copy of the query needed to be run on the database. The results are also reported in a single file (part-m-0000).

Multiple mappers can be used to process the query if the --split-by option is used. The split-by option is used to parallelize the SQL query. Each parallel task runs a subset of the main query, with the results of each sub-query being partitioned by bounding conditions inferred by Sqoop. Your query must include the token $CONDITIONS that each Sqoop process will replace with a unique condition expression based on the --split-by option. Note that $CONDITIONS is not an environment variable. Although Sqoop will try to create

balanced sub-queries based on the range of your primary key, it may be necessary to split on another column if your primary key is not uniformly distributed.

The following example illustrates the use of the --split-by option. First, remove the results of the previous query:

```
$ hdfs dfs -rm -r -skipTrash  sqoop-mysql-import/canada-city
```

Next, run the query using four mappers (-m 4), where we split by the ID number (--split-by ID):

```
sqoop  --options-file world-options.txt -m 4 --target-dir /user/hdfs/sqoop-mysql-import/canada-city --
query "SELECT ID,Name from City WHERE CountryCode='CAN' AND \$CONDITIONS" --split-by
ID
```

If we look at the number of results files, we find four files corresponding to the four mappers we requested in the command:

```
$ hdfs dfs -ls  sqoop-mysql-import/canada-city
Found 5 items
-rw-r--r--   2 hdfs hdfs       0 2014-08-18 21:31 sqoop-mysql-import/
canada-city/_SUCCESS
-rw-r--r--   2 hdfs hdfs     175 2014-08-18 21:31 sqoop-mysql-import/canada-city/
part-m-00000
-rw-r--r--   2 hdfs hdfs     153 2014-08-18 21:31 sqoop-mysql-import/canada-city/
part-m-00001
-rw-r--r--   2 hdfs hdfs     186 2014-08-18 21:31 sqoop-mysql-import/canada-city/
part-m-00002
-rw-r--r--   2 hdfs hdfs     182 2014-08-18 21:31 sqoop-mysql-import/canada-city/
part-m-00003
```

**Step 4: Export Data from HDFS to MySQL**

Sqoop can also be used to export data from HDFS. The first step is to create tables for exported data. There are actually two tables needed for each exported table. The first table holds the exported data (CityExport), and the second is used for staging the exported data (CityExportStaging). Enter the following MySQL commands to create these tables:

```
mysql> CREATE TABLE 'CityExport' (
       'ID' int(11) NOT NULL AUTO_INCREMENT,
       'Name' char(35) NOT NULL DEFAULT '',
       'CountryCode' char(3) NOT NULL DEFAULT '',
       'District' char(20) NOT NULL DEFAULT '',
       'Population' int(11) NOT NULL DEFAULT '0',
       PRIMARY KEY ('ID'));
mysql> CREATE TABLE 'CityExportStaging' (
       'ID' int(11) NOT NULL AUTO_INCREMENT,
       'Name' char(35) NOT NULL DEFAULT '',
       'CountryCode' char(3) NOT NULL DEFAULT '',
       'District' char(20) NOT NULL DEFAULT '',
       'Population' int(11) NOT NULL DEFAULT '0',
       PRIMARY KEY ('ID'));
```

Next, create a cities-export-options.txt file similar to the world-options.txt created previously, but use the export command instead of the import command.

The following command will export the cities data we previously imported back into MySQL:

**sqoop --options-file cities-export-options.txt --table CityExport --staging-table CityExportStaging --clear-staging-table -m 4 --export-dir /user/hdfs/sqoop-mysql-import/city**

Finally, to make sure everything worked correctly, check the table in MySQL to see if the cities are in the table:

```
$ mysql> select * from CityExport limit 10;
+----+----------------+-------------+---------------+------------+
| ID | Name           | CountryCode | District      | Population |
+----+----------------+-------------+---------------+------------+
|  1 | Kabul          | AFG         | Kabol         |    1780000 |
|  2 | Qandahar       | AFG         | Qandahar      |     237500 |
|  3 | Herat          | AFG         | Herat         |     186800 |
|  4 | Mazar-e-Sharif | AFG         | Balkh         |     127800 |
|  5 | Amsterdam      | NLD         | Noord-Holland |     731200 |
|  6 | Rotterdam      | NLD         | Zuid-Holland  |     593321 |
|  7 | Haag           | NLD         | Zuid-Holland  |     440900 |
|  8 | Utrecht        | NLD         | Utrecht       |     234323 |
|  9 | Eindhoven      | NLD         | Noord-Brabant |     201843 |
| 10 | Tilburg        | NLD         | Noord-Brabant |     193238 |
+----+----------------+-------------+---------------+------------+
10 rows in set (0.00 sec)
```

### Some Handy Cleanup Commands

If you are not especially familiar with MySQL, the following commands may be helpful to clean up the examples. To remove the table in MySQL, enter the following command:

mysql> **drop table 'CityExportStaging';**

To remove the data in a table, enter this command:

mysql> **delete from CityExportStaging;**

To clean up imported files, enter this command:

$ **hdfs dfs -rm -r -skipTrash sqoop-mysql-import/{country,city, canada-city}**

## USING APACHE FLUME TO ACQUIRE DATA STREAMS

Apache Flume is an independent agent designed to collect, transport, and store data into HDFS. Often data transport involves a number of Flume agents that may traverse a series of machines and locations. Flume is often used for log files, social media-generated data, email messages, and just about any continuous data source. As shown in Figure 7.3, a Flume agent is composed of three components.

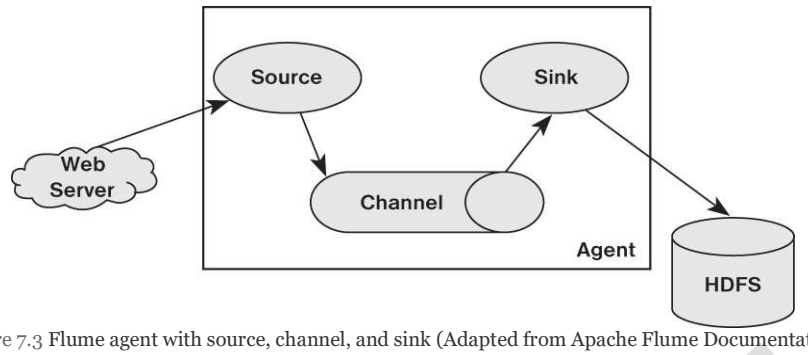



Figure 7.3 Flume agent with source, channel, and sink (Adapted from Apache Flume Documentation)

■ *Source.* The source component receives data and sends it to a channel. It can send the data to more than one channel. The input data can be from a real-time source (e.g., weblog) or another Flume agent.

■ *Channel.* A channel is a data queue that forwards the source data to the sink destination. It can be thought of as a buffer that manages input (source) and output (sink) flow rates.

■ *Sink.* The sink delivers data to destination such as HDFS, a local file, or another Flume agent.

A Flume agent must have all three of these components defined. A Flume agent can have several sources, channels, and sinks. Sources can write to multiple channels, but a sink can take data from only a single channel. Data written to a channel remain in the channel until a sink removes the data. By default, the data in a channel are kept in memory but may be optionally stored on disk to prevent data loss in the event of a network failure.

As shown in Figure 7.4, Sqoop agents may be placed in a pipeline, possibly to traverse several machines or domains. This configuration is normally used when data are collected on one machine (e.g., a web server) and sent to another machine that has access to HDFS.

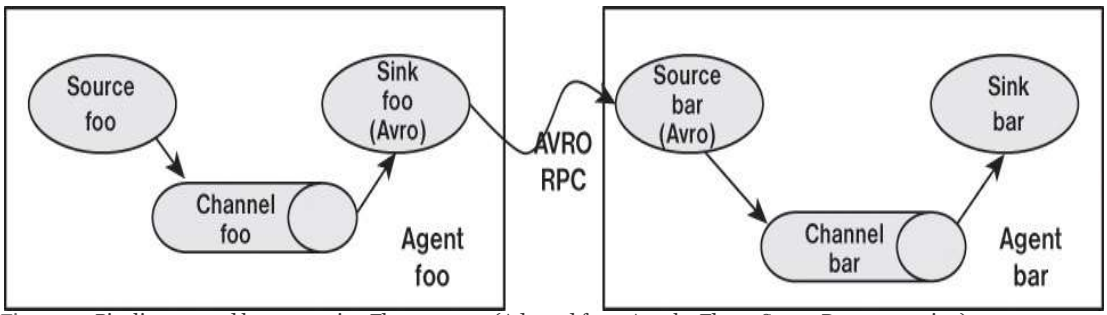



Figure 7.4 Pipeline created by connecting Flume agents (Adapted from Apache Flume Sqoop Documentation)

In a Flume pipeline, the sink from one agent is connected to the source of another. The data transfer format normally used by Flume, which is called Apache Avro, provides several useful features. First, Avro is a data serialization/deserialization system that uses a compact

binary format. The schema is sent as part of the data exchange and is defined using JSON (JavaScript Object Notation). Avro also uses remote procedure calls (RPCs) to send data. That is, an Avro sink will contact an Avro source to send data.

Another useful Flume configuration is shown in Figure 7.5. In this configuration, Flume is used to consolidate several data sources before committing them to HDFS.

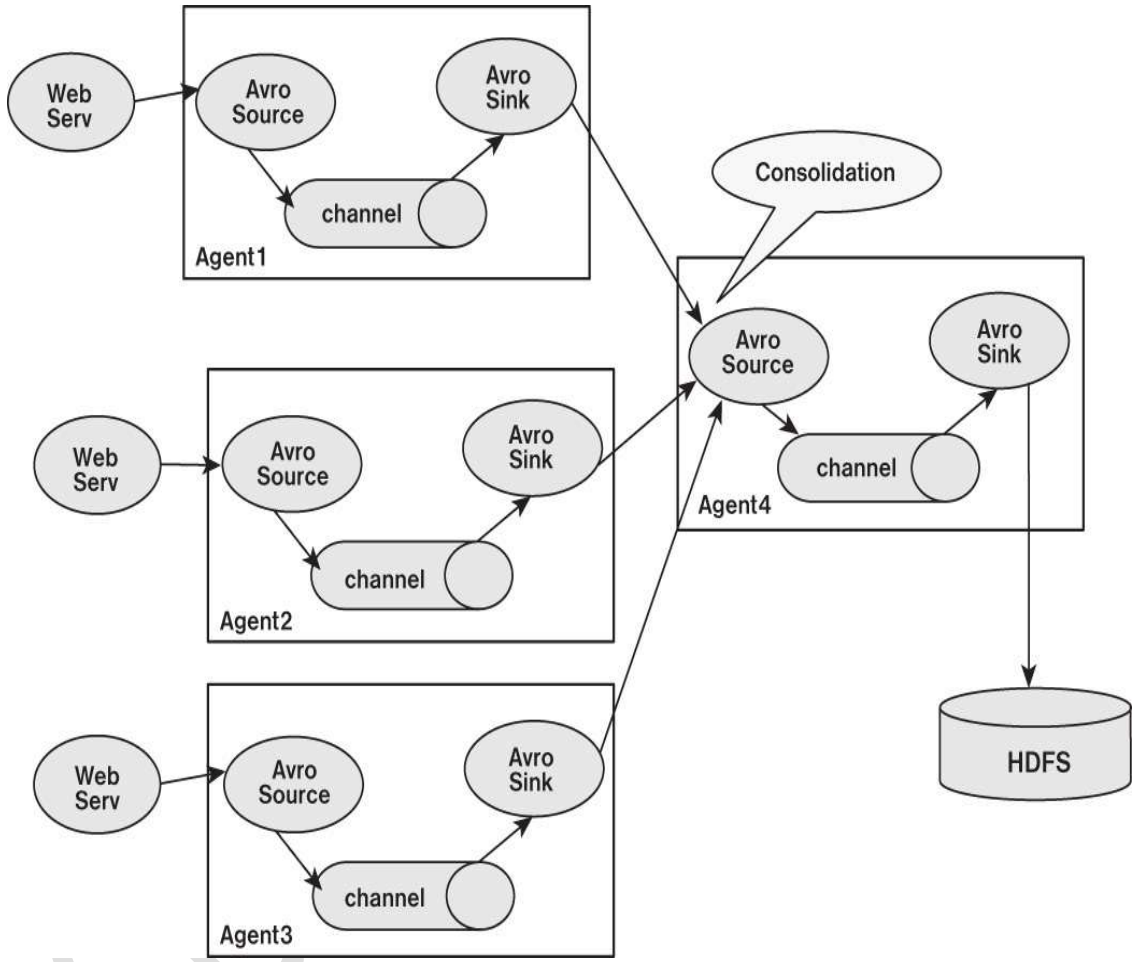



Figure 7.5 A Flume consolidation network (Adapted from Apache Flume Documentation)

There are many possible ways to construct Flume transport networks. In addition, other Flume features not described in depth here include plug-ins and interceptors that can enhance Flume pipelines.

### Flume Example Walk-Through

Follow these steps to walk through a Flume example.

**Step 1: Download and Install Apache Flume**

**Step 2: Simple Test**

A simple test of Flume can be done on a single machine. To start the Flume agent, enter the flume-ng command shown here. This command uses the simple-example.conf file to configure the agent.

```
$ flume-ng agent --conf conf --conf-file simple-example.conf --name simple_agent -
Dflume.root.logger=INFO,console
```

In another terminal window, use telnet to contact the agent:

```
$ telnet localhost 44444
  Trying ::1...
  telnet: connect to address ::1: Connection refused
  Trying 127.0.0.1...
  Connected to localhost.
  Escape character is '^]'.
  testing  1 2 3
  OK
```

If Flume is working correctly, the window where the Flume agent was started will show the testing message entered in the telnet window:

**Step 3: Weblog Example**

In this example, a record from the weblogs from the local machine (Ambari output) will be placed into HDFS using Flume. This example is easily modified to use other weblogs from different machines. Two files are needed to configure Flume. (See the sidebar and Appendix A for file downloading instructions.)

- `web-server-target-agent.conf`—the target Flume agent that writes the data to HDFS
- `web-server-source-agent.conf`—the source Flume agent that captures the weblog data

The weblog is also mirrored on the local file system by the agent that writes to HDFS. To run the example, create the directory as root:

```
# mkdir /var/log/flume-hdfs
# chown hdfs:hadoop /var/log/flume-hdfs/
```

Next, as user hdfs, make a Flume data directory in HDFS:

```
$ hdfs dfs -mkdir /user/hdfs/flume-channel/
```

Now that you have created the data directories, you can start the Flume target agent (execute as user hdfs):

```
$ flume-ng agent -c conf -f web-server-target-agent.conf -n collector
```

This agent writes the data into HDFS and should be started before the source agent. (The source reads the weblogs.) This configuration enables automatic use of the Flume agent. The /etc/flume/conf/{flume.conf, flume-env.sh.template} files need to be configured for this purpose. For this example, the /etc/flume/conf/flume.conf file can be the same as the web-server-target.conf file (modified for your environment).

In this example, the source agent is started as root, which will start to feed the weblog data to the target agent. Alternatively, the source agent can be on another machine if desired.

# **flume-ng agent -c conf -f web-server-source-agent.conf -n source_agent**

To see if Flume is working correctly, check the local log by using the tail command. Also confirm that the flume-ng agents are not reporting any errors (the file name will vary).

$ **tail -f /var/log/flume-hdfs/1430164482581-1**

The contents of the local log under flume-hdfs should be identical to that written into HDFS. You can inspect this file by using the hdfs -tail command (the file name will vary). Note that while running Flume, the most recent file in HDFS may have the extension .tmp appended to it. The .tmpindicates that the file is still being written by Flume. The target agent can be configured to write the file (and start another .tmp file) by setting some or all of the rollCount, rollSize, rollInterval, idleTimeout, and batchSize options in the configuration file.

$ hdfs dfs -tail flume-channel/apache_access_combined/150427/FlumeData.1430164801381

Both files should contain the same data. For instance, the preceding example had the following data in both files:

```
10.0.0.1 - - [27/Apr/2015:16:04:21 -0400] "GET /ambarinagios/nagios/
nagios_alerts.php?q1=alerts&alert_type=all HTTP/1.1" 200 30801 "-" "Java/1.7.0_65"
10.0.0.1 - - [27/Apr/2015:16:04:25 -0400] "POST /cgi-bin/rrd.py HTTP/1.1" 200 784
"-" "Java/1.7.0_65"
10.0.0.1 - - [27/Apr/2015:16:04:25 -0400] "POST /cgi-bin/rrd.py HTTP/1.1" 200 508
"-" "Java/1.7.0_65"
```

## MANAGE HADOOP WORKFLOWS WITH APACHE OOZIE

Oozie is a workflow director system designed to run and manage multiple related Apache Hadoop jobs. For instance, complete data input and analysis may require several discrete Hadoop jobs to be run as a workflow in which the output of one job serves as the input for a successive job. Oozie is designed to construct and manage these workflows. Oozie is not a substitute for the YARN scheduler. That is, YARN manages resources for individual Hadoop jobs, and Oozie provides a way to connect and control Hadoop jobs on the cluster.

Oozie workflow jobs are represented as directed acyclic graphs (DAGs) of actions. (DAGs are basically graphs that cannot have directed loops.) Three types of Oozie jobs are permitted:

■ Workflow—a specified sequence of Hadoop jobs with outcome-based decision points and control dependency. Progress from one action to another cannot happen until the first action is complete.

■ Coordinator—a scheduled workflow job that can run at various time intervals or when data become available.

■ Bundle—a higher-level Oozie abstraction that will batch a set of coordinator jobs.

Oozie is integrated with the rest of the Hadoop stack, supporting several types of Hadoop jobs out of the box (e.g., Java MapReduce, Streaming MapReduce, Pig, Hive, and Sqoop) as well as system-specific jobs (e.g., Java programs and shell scripts). Oozie also provides a CLI and a web UI for monitoring jobs.

Figure 7.6 depicts a simple Oozie workflow. In this case, Oozie runs a basic MapReduce operation. If the application was successful, the job ends; if an error occurred, the job is killed.

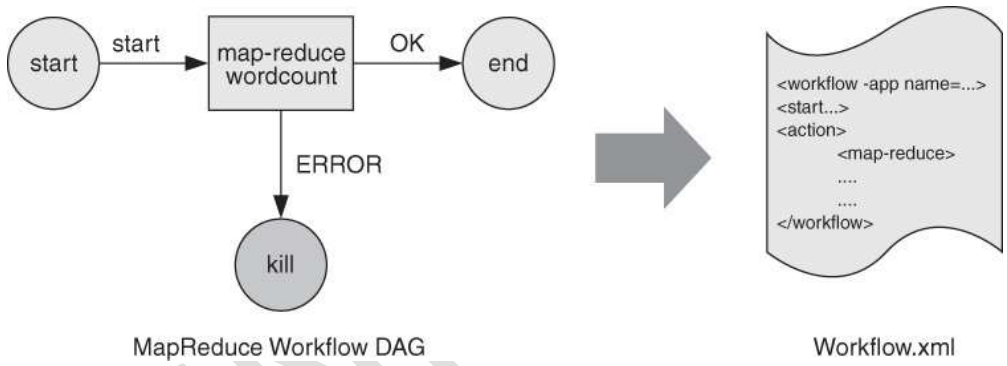



Figure 7.6 A simple Oozie DAG workflow (Adapted from Apache Oozie Documentation)

Oozie workflow definitions are written in hPDL (an XML Process Definition Language). Such workflows contain several types of nodes:

■ **Control flow nodes** define the beginning and the end of a workflow. They include start, end, and optional fail nodes.

■ **Action nodes** are where the actual processing tasks are defined. When an action node finishes, the remote systems notify Oozie and the next node in the workflow is executed. Action nodes can also include HDFS commands.

■ **Fork/join nodes** enable parallel execution of tasks in the workflow. The fork node enables two or more tasks to run at the same time. A join node represents a rendezvous point that must wait until all forked tasks complete.

■ **Control flow nodes** enable decisions to be made about the previous task. Control decisions are based on the results of the previous action (e.g., file size or file existence). Decision nodes are essentially switch-case statements that use JSP EL (Java Server Pages—Expression Language) that evaluate to either true or false.

Figure 7.7 depicts a more complex workflow that uses all of these node types.

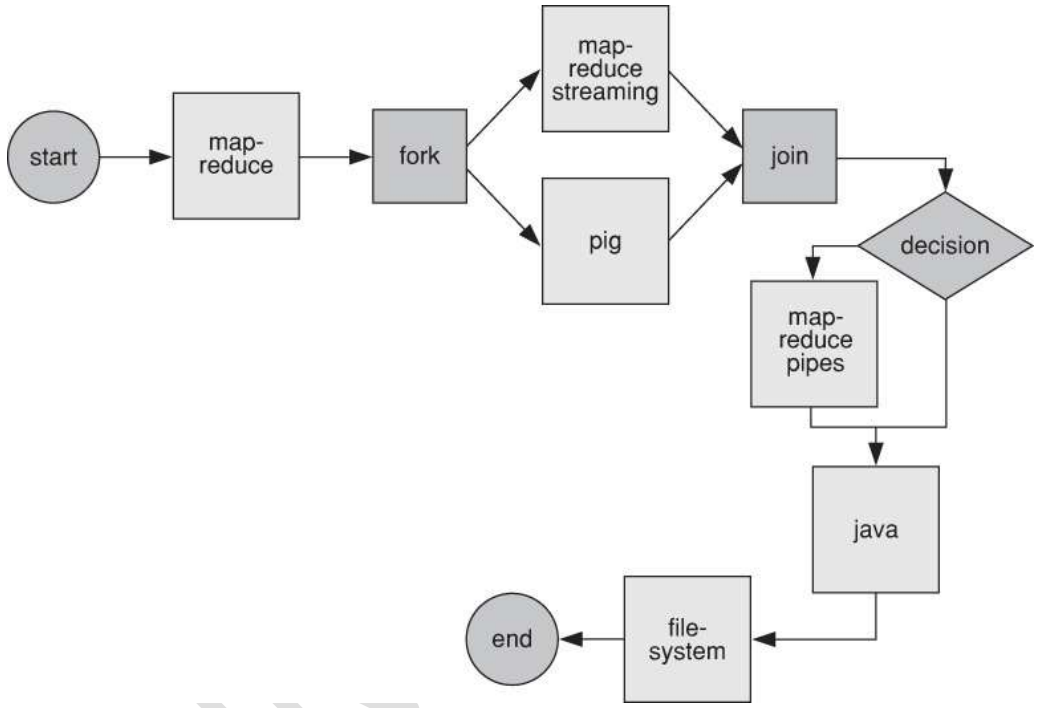



Figure 7.7 A more complex Oozie DAG workflow (Adapted from Apache Oozie Documentation)

### Oozie Example Walk-Through

#### Step 1: Download Oozie Examples

The Oozie examples used in this section can be found on the book website (see Appendix A). They are also available as part of the oozie-client.noarch RPM in the Hortonworks HDP 2.x packages. For HDP 2.1, the following command can be used to extract the files into the working directory used for the demo:

$ **tar xvzf /usr/share/doc/oozie-4.0.0.2.1.2.1/oozie-examples.tar.gz**

For HDP 2.2, the following command will extract the files:
$ **tar xvzf /usr/hdp/2.2.4.2-2/oozie/doc/oozie-examples.tar.gz**

Once extracted, rename the examples directory to oozie-examples so that you will not confuse it with the other examples directories.

$ **mv examples oozie-examples**

The examples must also be placed in HDFS. Enter the following command to move the example files into HDFS:

$ **hdfs dfs -put oozie-examples/ oozie-examples**

The Oozie shared library must be installed in HDFS. If you are using the Ambari installation of HDP 2.x, this library is already found in HDFS: /user/oozie/share/lib.

**Step 2: Run the Simple MapReduce Example**

Move to the simple MapReduce example directory:

$ **cd oozie-examples/apps/map-reduce/**

This directory contains two files and a lib directory. The files are:

■ The job.properties file defines parameters (e.g., path names, ports) for a job. This file may change per job.

■ The workflow.xml file provides the actual workflow for the job. In this case, it is a simple MapReduce (pass/fail). This file usually stays the same between jobs.

The job.properties file included in the examples requires a few edits to work properly. Using a text editor, change the following lines by adding the host name of the NameNode and ResourceManager (indicated by jobTracker in the file).

As shown in Figure 7.6, this simple workflow runs an example MapReduce job and prints an error message if it fails.

To run the Oozie MapReduce example job from the oozie-examples/apps/map-reduce directory, enter the following line:

$ **oozie job -run -oozie http://limulus:11000/oozie -config job.properties**

When Oozie accepts the job, a job ID will be printed:

job: 0000001-150424174853048-oozie-oozi-W

You will need to change the "limulus" host name to match the name of the node running your Oozie server. The job ID can be used to track and control job progress.

To avoid having to provide the -oozie option with the Oozie URL every time you run the ooziecommand, set the OOZIE_URL environment variable as follows (using your Oozie server host name in place of "limulus"):

$ **export OOZIE_URL="http://limulus:11000/oozie"**

You can now run all subsequent Oozie commands without specifying the -oozie URL option. For instance, using the job ID, you can learn about a particular job's progress by issuing the following command:

$ **oozie job -info 0000001-150424174853048-oozie-oozi-W**

The resulting output (line length compressed) is shown in the following listing. Because this job is just a simple test, it may be complete by the time you issue the -info command. If it is not complete, its progress will be indicated in the listing.

```
Job ID : 0000001-150424174853048-oozie-oozi-W
------------------------------------------------------------------------------
Workflow Name : map-reduce-wf
App Path      : hdfs://limulus:8020/user/hdfs/examples/apps/map-reduce
Status        : SUCCEEDED
Run           : 0
User          : hdfs
Group         : -
Created       : 2015-04-29 20:52 GMT
Started       : 2015-04-29 20:52 GMT
Last Modified : 2015-04-29 20:53 GMT
Ended         : 2015-04-29 20:53 GMT
CoordAction ID: -

Actions
------------------------------------------------------------------------------
ID                                Status   Ext ID    Ext Status  Err Code
------------------------------------------------------------------------------
0000001-150424174853048-oozie
  -oozi-W@:start:                 OK       -         OK          -
------------------------------------------------------------------------------
0000001-150424174853048-oozie
  -oozi-W@mr-node                 OK  job_1429912013449_0006 SUCCEEDED -
------------------------------------------------------------------------------
0000001-150424174853048-oozie
  -oozi-W@end                     OK       -         OK          -
------------------------------------------------------------------------------
```

The various steps shown in the output can be related directly to the workflow.xml mentioned previously. Note that the MapReduce job number is provided. This job will also be listed in the ResourceManager web user interface. The application output is located in HDFS under the oozie-examples/output-data/map-reduce directory.

**Step 3: Run the Oozie Demo Application**

A more sophisticated example can be found in the demo directory (oozie-examples/apps/demo). This workflow includes MapReduce, Pig, and file system tasks as well as fork, join, decision, action, start, stop, kill, and end nodes.

Move to the demo directory and edit the job.properties file as described previously. Entering the following command runs the workflow (assuming the OOZIE_URL environment variable has been set):

$ **oozie job -run -config job.properties**

You can track the job using either the Oozie command-line interface or the Oozie web console. To start the web console from within Ambari, click on the Oozie service, and then click on the Quick Links pull-down menu and select Oozie Web UI. Alternatively, you can start the Oozie web UI by connecting to the Oozie server directly. For example, the following command will bring up the Oozie UI (use your Oozie server host name in place of "limulus"):

$ **firefox http://limulus:11000/oozie/**

Figure 7.8 shows the main Oozie console window.

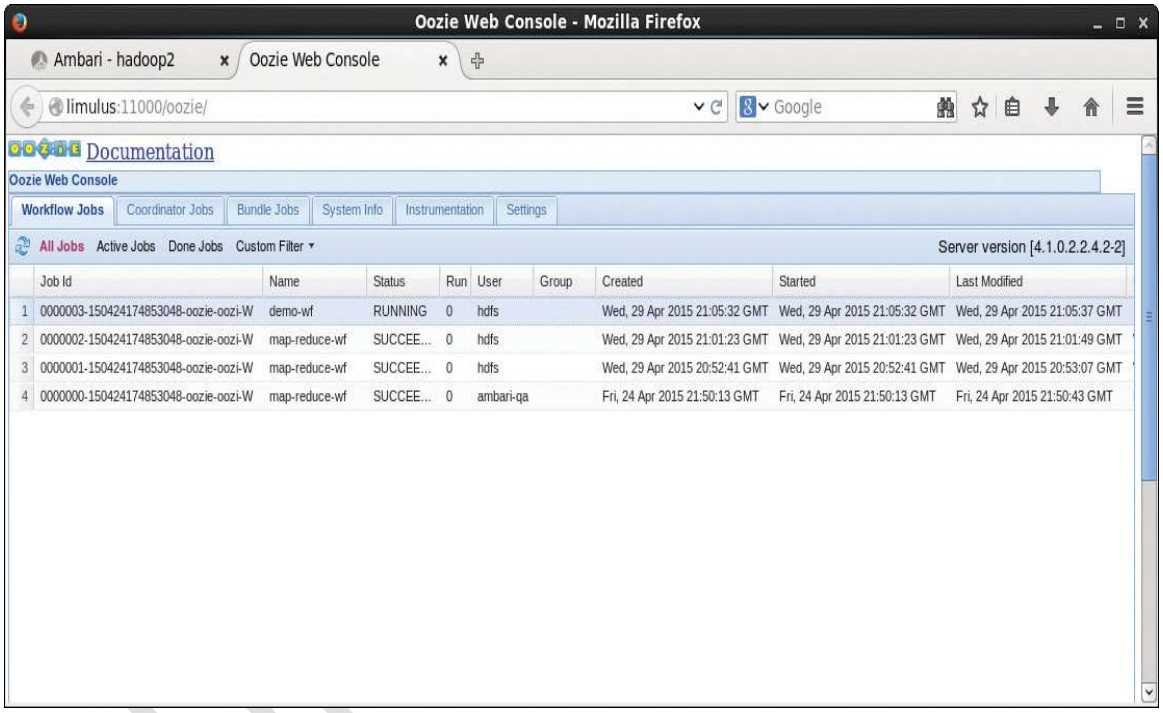



Figure 7.8 Oozie main console window

Workflow jobs are listed in tabular form, with the most recent job appearing first. If you click on a workflow, the Job Info window in Figure 7.9 will be displayed. The job progression results, similar to those printed by the Oozie command line, are shown in the Actions window at the bottom.

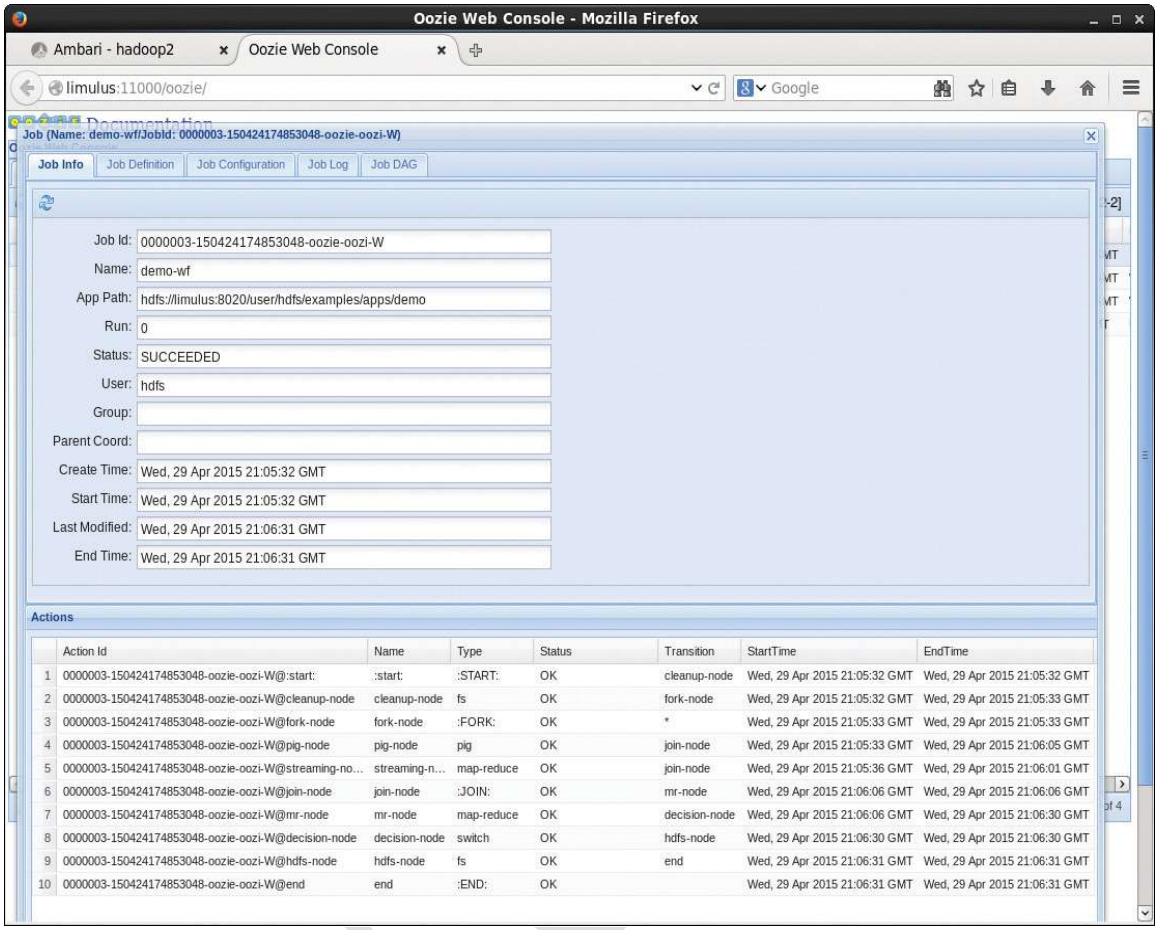



Figure 7.9 Oozie workflow information window

Other aspects of the job can be examined by clicking the other tabs in the window. The last tab actually provides a graphical representation of the workflow DAG. If the job is not complete, it will highlight the steps that have been completed thus far. The DAG for the demo example is shown in Figure 7.10. The actual image was split to fit better on the page. As with the previous example, comparing this information to workflow.xml file can provide further insights into how Oozie operates.

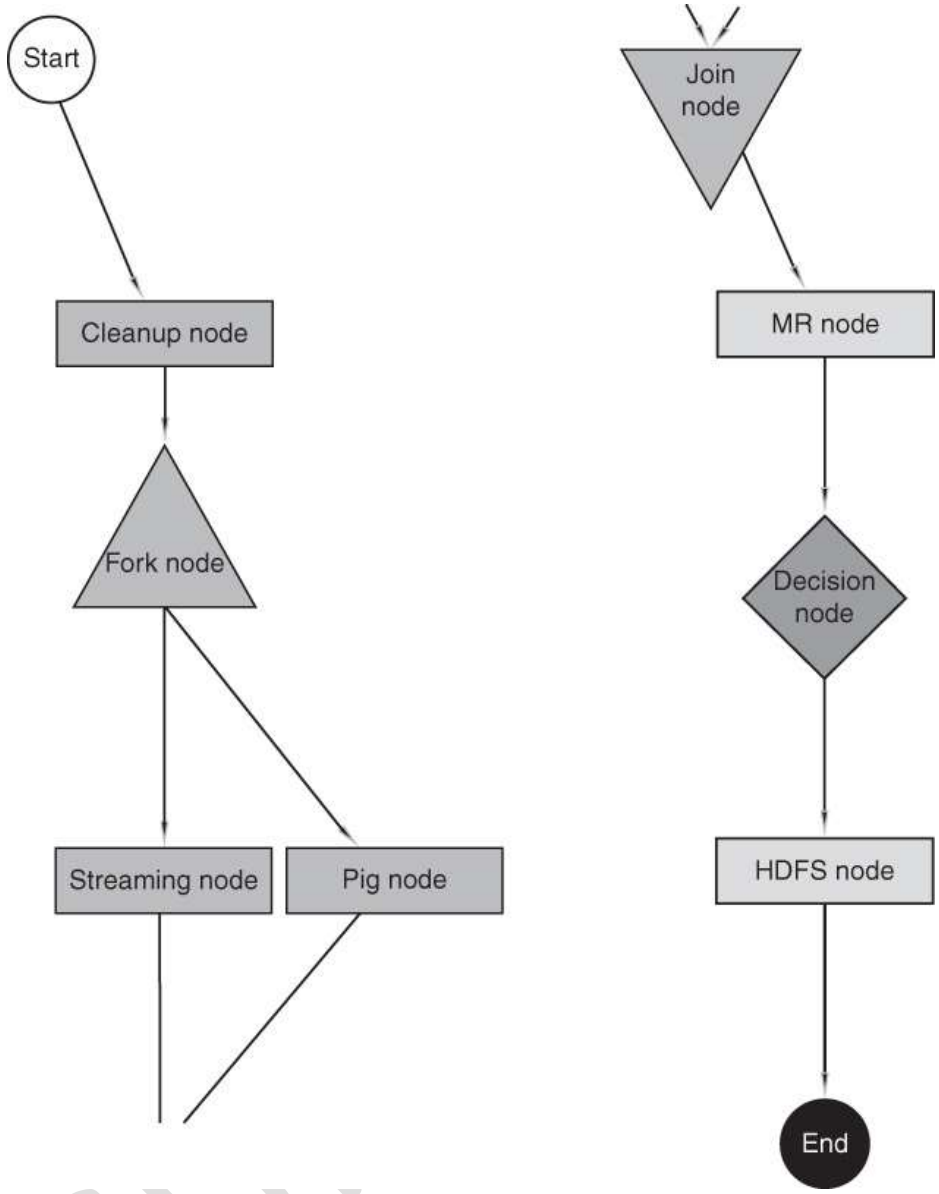



Figure 7.10 Oozie-generated workflow DAG for the demo example, as it appears on the screen

**A Short Summary of Oozie Job Commands**

The following summary lists some of the more commonly encountered Oozie commands. See the latest documentation at http://oozie.apache.org for more information. (Note that the examples here assume OOZIE_URL is defined.)

■ Run a workflow job (returns `_OOZIE_JOB_ID_`):

```
$ oozie job -run -config JOB_PROPERITES
```

■ Submit a workflow job (returns `_OOZIE_JOB_ID_` but does not start):

```
$ oozie job -submit -config JOB_PROPERTIES
```

- Start a submitted job:

```
$ oozie job -start _OOZIE_JOB_ID_
```

- Check a job's status:

```
$ oozie job -info _OOZIE_JOB_ID_
```

- Suspend a workflow:

```
$ oozie job -suspend _OOZIE_JOB_ID_
```

- Resume a workflow:

```
$ oozie job -resume _OOZIE_JOB_ID_
```

- Rerun a workflow:

```
$ oozie job -rerun _OOZIE_JOB_ID_ -config JOB_PROPERTIES
```

- Kill a job:

```
$ oozie job -kill _OOZIE_JOB_ID_
```

- View server logs:

```
$ oozie job  -logs _OOZIE_JOB_ID_
```

Full logs are available at /var/log/oozie on the Oozie server.

## USING APACHE HBASE

Apache HBase is an open source, distributed, versioned, nonrelational database modeled after Google's Bigtable. Like Bigtable, HBase leverages the distributed data storage provided by the underlying distributed file systems spread across commodity servers. Apache HBase provides Bigtable-like capabilities on top of Hadoop and HDFS. Some of the more important features include the following capabilities:

- Linear and modular scalability
- Strictly consistent reads and writes
- Automatic and configurable sharding of tables
- Automatic failover support between RegionServers
- Convenient base classes for backing Hadoop MapReduce jobs with Apache HBase tables
- Easy-to-use Java API for client access

### HBase Data Model Overview

A table in HBase is similar to other databases, having rows and columns. Columns in HBase are grouped into column families, all with the same prefix. For example, consider a table of daily stock prices. There may be a column family called “price” that has four members—price:open, price:close, price:low, and price:high. A column does not need to be a family. For instance, the stock table may have a column named “volume” indicating how many shares were traded. All column family members are stored together in the physical file system.

Specific HBase cell values are identified by a row key, column (column family and column), and version (timestamp). It is possible to have many versions of data within an HBase cell. A version is specified as a timestamp and is created each time data are written to a cell. Almost anything can serve as a row key, from strings to binary representations of longs to serialized data structures. Rows are lexicographically sorted with the lowest order appearing first in a table. The empty byte array denotes both the start and the end of a table’s namespace. All table accesses are via the table row key, which is considered its primary key.

### HBase Example Walk-Through

HBase provides a shell for interactive use. To enter the shell, type the following as a user:

```
$ hbase shell
hbase(main):001:0>
```

To exit the shell, type **exit**.

Various commands can be conveniently entered from the shell prompt. For instance, the status command provides the system status:

```
hbase(main):001:0> status
4 servers, 0 dead, 1.0000 average load
```

Additional arguments can be added to the status command, including 'simple', 'summary', or 'detailed'. The single quotes are needed for proper operation. For example, the following command will provide simple status information for the four HBase servers (actual server statistics have been removed for clarity):

```
hbase(main):002:0> status 'simple'
4 live servers
   n1:60020 1429912048329
      ...
   n2:60020 1429912040653
      ...
   limulus:60020 1429912041396
      ...
   n0:60020 1429912042885
      ...
0 dead servers
Aggregate load: 0, regions: 4
```

Other basic commands, such as `version` or `whoami`, can be entered directly at the hbase(main)prompt. In the example that follows, we will use a small set of daily stock price data for Apple computer. The data have the following form:

| Date | Open | High | Low | Close | Volume |
|---|---|---|---|---|---|
| 6-May-15 | 126.56 | 126.75 | 123.36 | 125.01 | 71820387 |

The data can be downloaded from Google using the following command. Note that other stock prices are available by changing the NASDAQ:AAPL argument to any other valid exchange and stock name (e.g., NYSE: IBM).

**$ wget -O Apple-stock.csv**
**http://www.google.com/finance/historical?q=NASDAQ:AAPL\&authuser=0\&output=csv**

The Apple stock price database is in comma-separated format (csv) and will be used to illustrate some basic operations in the HBase shell.

### Create the Database

The next step is to create the database in HBase using the following command:

```
hbase(main):006:0> create 'apple', 'price' , 'volume'
0 row(s) in 0.8150 seconds
```

In this case, the table name is apple, and two columns are defined. The date will be used as the row key. The price column is a family of four values (open, close, low, high). The put command is used to add data to the database from within the shell. For instance, the preceding data can be entered by using the following commands:

```
put 'apple','6-May-15','price:open','126.56'
put 'apple','6-May-15','price:high','126.75'
put 'apple','6-May-15','price:low','123.36'
put 'apple','6-May-15','price:close','125.01'
put 'apple','6-May-15','volume','71820387'
```

The shell also keeps a history for the session, and previous commands can be retrieved and edited for resubmission.

### Inspect the Database

The entire database can be listed using the scan command. Be careful when using this command with large databases. This example is for one row.

```
hbase(main):006:0> scan 'apple'
ROW          COLUMN+CELL
 6-May-15     column=price:close, timestamp=1430955128359, value=125.01
 6-May-15     column=price:high, timestamp=1430955126024, value=126.75
 6-May-15     column=price:low, timestamp=1430955126053, value=123.36
 6-May-15     column=price:open, timestamp=1430955125977, value=126.56
 6-May-15     column=volume:, timestamp=1430955141440, value=71820387
```

### Get a Row

You can use the row key to access an individual row. In the stock price database, the date is the row key.

```
hbase(main):008:0> get 'apple', '6-May-15'
COLUMN                  CELL
 price:close            timestamp=1430955128359, value=125.01
 price:high             timestamp=1430955126024, value=126.75
 price:low              timestamp=1430955126053, value=123.36
 price:open             timestamp=1430955125977, value=126.56
 volume:                timestamp=1430955141440, value=71820387
5 row(s) in 0.0130 seconds
```

### Get Table Cells

A single cell can be accessed using the get command and the COLUMN option:

```
hbase(main):013:0> get 'apple', '5-May-15', {COLUMN => 'price:low'}
COLUMN                  CELL
 price:low              timestamp=1431020767444, value=125.78
1 row(s) in 0.0080 seconds
```

In a similar fashion, multiple columns can be accessed as follows:

```
hbase(main):012:0> get 'apple', '5-May-15', {COLUMN => ['price:low', 'price:high']}
COLUMN                  CELL
 price:high             timestamp=1431020767444, value=128.45
 price:low              timestamp=1431020767444, value=125.78
2 row(s) in 0.0070 seconds
```

### Delete a Cell

A specific cell can be deleted using the following command:

```
hbase(main):009:0> delete 'apple', '6-May-15' , 'price:low'
```

If the row is inspected using get, the price:low cell is not listed.

```
hbase(main):010:0> get 'apple', '6-May-15'
COLUMN                  CELL
 price:close            timestamp=1430955128359, value=125.01
 price:high             timestamp=1430955126024, value=126.75
 price:open             timestamp=1430955125977, value=126.46
 volume:                timestamp=1430955141440, value=71820387
4 row(s) in 0.0130 seconds
```

### Delete a Row

You can delete an entire row by giving the deleteall command as follows:

```
hbase(main):009:0> deleteall 'apple', '6-May-15'
```

### Remove a Table

To remove (drop) a table, you must first disable it. The following two commands remove the appletable from Hbase:

```
hbase(main):009:0> disable 'apple'
hbase(main):010:0> drop 'apple'
```

**Scripting Input**

Commands to the HBase shell can be placed in bash scripts for automated processing. For instance, the following can be placed in a bash script:

```
echo "put 'apple','6-May-15','price:open','126.56'" | hbase shell
```

The book software page includes a script (input_to_hbase.sh) that imports the Apple-stock.csv file into HBase using this method. It also removes the column titles in the first line. The script will load the entire file into HBase when you issue the following command:

```
$ input_to_hbase.sh Apple-stock.csv
```

While the script can be easily modified to accommodate other types of data, it is not recommended for production use because the upload is very inefficient and slow. Instead, this script is best used to experiment with small data files and different types of data.

**Adding Data in Bulk**

There are several ways to efficiently load bulk data into HBase. Covering all of these methods is beyond the scope of this chapter. Instead, we will focus on the ImportTsv utility, which loads data in tab-separated values (tsv) format into HBase. It has two distinct usage modes:

- Loading data from a tsv-format file in HDFS into HBase via the put command
- Preparing StoreFiles to be loaded via the completebulkload utility

The following example shows how to use ImportTsv for the first option, loading the tsv-format file using the put command.

The first step is to convert the Apple-stock.csv file to tsv format. The following script, which is included in the book software, will remove the first line and do the conversion. In doing so, it creates a file named Apple-stock.tsv.

```
$ convert-to-tsv.sh Apple-stock.csv
```

Next, the new file is copied to HDFS as follows:

```
$ hdfs dfs -put Apple-stock.tsv /tmp
```

Finally, ImportTsv is run using the following command line. Note the column designation in the -Dimporttsv.columns option. In the example, the HBASE_ROW_KEY is set as the first column—that is, the date for the data.

```
$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -
Dimporttsv.columns=HBASE_ROW_KEY,price:open,price:high,price:low,price:close,volume apple
/tmp/Apple-stock.tsv
```

The ImportTsv command will use MapReduce to load the data into HBase. To verify that the command works, drop and re-create the apple database, as described previously, before running the import command.

## 8. Hadoop YARN Applications

**In This Chapter:**

- The YARN Distributed-Shell is introduced as a non-MapReduce application.
- The Hadoop YARN application and operation structure is explained.
- A summary of YARN application frameworks is provided.

### YARN DISTRIBUTED-SHELL

The Hadoop YARN project includes the Distributed-Shell application, which is an example of a Hadoop non-MapReduce application built on top of YARN. Distributed-Shell is a simple mechanism for running shell commands and scripts in containers on multiple nodes in a Hadoop cluster. This application is not meant to be a production administration tool, but rather a demonstration of the non-MapReduce capability that can be implemented on top of YARN. There are multiple mature implementations of a distributed shell that administrators typically use to manage a cluster of machines.

In addition, Distributed-Shell can be used as a starting point for exploring and building Hadoop YARN applications. This chapter offers guidance on how the Distributed-Shell can be used to understand the operation of YARN applications.

### USING THE YARN DISTRIBUTED-SHELL

For the purpose of the examples presented in the remainder of this chapter, we assume and assign the following installation path, based on Hortonworks HDP 2.2, the Distributed-Shell application:

```
$ export YARN_DS=/usr/hdp/current/hadoop-yarn-client/hadoop-yarn-applications-
distributedshell.jar
```

For the pseudo-distributed install using Apache Hadoop version 2.6.0, the following path will run the Distributed-Shell application (assuming $HADOOP_HOME is defined to reflect the location Hadoop):

$ **export YARN_DS=$HADOOP_HOME/share/hadoop/yarn/hadoop-yarn-applications-distributedshell-2.6.0.jar**

If another distribution is used, search for the file hadoop-yarn-applications-distributedshell*.jar and set $YARN_DS based on its location. Distributed-Shell exposes various options that can be found by running the following command:

$ **yarn org.apache.hadoop.yarn.applications.distributedshell.Client -jar $YARN_DS -help**

The output of this command follows; we will explore some of these options in the examples illustrated in this chapter.

```
usage: Client
 -appname <arg>                Application Name. Default value – DistributedShell
 -container_memory <arg>       Amount of memory in MB to be requested to run the shell command
 -container_vcores <arg>       Amount of virtual cores to be requested to run the shell command
 -create                       Flag to indicate whether to create the domain specified with -domain.
 -debug                        Dump out debug information
 -domain <arg>                 ID of the timeline domain where the timeline entities will be put
 -help                         Print usage
 -jar <arg>                    Jar file containing the application master
 -log_properties <arg>         log4j.properties file
 -master_memory <arg>          Amount of memory in MB to be requested to run the application master
 -master_vcores <arg>          Amount of virtual cores to be requested to run the application master
 -modify_acls <arg>            Users and groups that allowed to modify the timeline entities in the given domain
 -timeout <arg>                Application timeout in milliseconds
 -view_acls <arg>              Users and groups that allowed to view the timeline entities in the  given domain
```

### A Simple Example

The simplest use-case for the Distributed-Shell application is to run an arbitrary shell command in a container. We will demonstrate the use of the uptime command as an example. This command is run on the cluster using Distributed-Shell as follows:

$ **yarn org.apache.hadoop.yarn.applications.distributedshell.Client -jar $YARN_DS -shell_command uptime**

By default, Distributed-Shell spawns only one instance of a given shell command. When this command is run, you can see progress messages on the screen but nothing about the actual shell command. If the shell command succeeds, the following should appear at the end of the output:

```
15/05/27 14:48:53 INFO distributedshell.Client: Application completed successfully
```

If the shell command did not work for whatever reason, the following message will be displayed:

```
15/05/27 14:58:42 ERROR distributedshell.Client: Application failed to complete
successfully
```

The next step is to examine the output for the application. Distributed-Shell redirects the output of the individual shell commands run on the cluster nodes into the log files, which are found either on the individual nodes or aggregated onto HDFS, depending on whether log aggregation is enabled.

Assuming log aggregation is enabled, the results for each instance of the command can be found by using the yarn logs command. For the previous uptime example, the following command can be used to inspect the logs:

```
$ yarn logs -applicationId application_1432831236474_0001
```

The abbreviated output follows:

```
Container: container_1432831236474_0001_01_000001 on n0_45454
==========================================================
LogType:AppMaster.stderr
Log Upload Time:Thu May 28 12:41:58 -0400 2015
LogLength:3595
Log Contents:
15/05/28 12:41:52 INFO distributedshell.ApplicationMaster: Initializing
ApplicationMaster
[...]
Container: container_1432831236474_0001_01_000002 on n1_45454
==========================================================
LogType:stderr
Log Upload Time:Thu May 28 12:41:59 -0400 2015
LogLength:0
Log Contents:

LogType:stdout
Log Upload Time:Thu May 28 12:41:59 -0400 2015
LogLength:71
Log Contents:
 12:41:56 up 33 days, 19:28,  0 users,  load average: 0.08, 0.06, 0.01
```

Notice that there are two containers. The first container (`con..._000001`) is the ApplicationMaster for the job. The second container (con..._000002) is the actual shell script. The output for the uptime command is located in the second containers stdout after the Log Contents: label.

**Using More Containers**

Distributed-Shell can run commands to be executed on any number of containers by way of the -num_containers argument. For example, to see on which nodes the Distributed-Shell command was run, the following command can be used:

$ **yarn org.apache.hadoop.yarn.applications.distributedshell.Client -jar $YARN_DS -shell_command hostname -num_containers 4**

If we now examine the results for this job, there will be five containers in the log. The four command containers (2 through 5) will print the name of the node on which the container was run.

### Distributed-Shell Examples with Shell Arguments

Arguments can be added to the shell command using the -shell_args option. For example, to do a ls -l in the directory from where the shell command was run, we can use the following commands:

$ **yarn org.apache.hadoop.yarn.applications.distributedshell.Client -jar $YARN_DS -shell_command ls -shell_args -l**

The resulting output from the log file is as follows:

```
total 20
-rw-r--r-- 1 yarn hadoop   74 May 28 10:37 container_tokens
-rwx------ 1 yarn hadoop  643 May 28 10:37 default_container_executor_session.sh
-rwx------ 1 yarn hadoop  697 May 28 10:37 default_container_executor.sh
-rwx------ 1 yarn hadoop 1700 May 28 10:37 launch_container.sh
drwx--x--- 2 yarn hadoop 4096 May 28 10:37 tmp
```

As can be seen, the resulting files are new and not located anywhere in HDFS or the local file system. When we explore further by giving a pwd command for Distributed-Shell, the following directory is listed and created on the node that ran the shell command:

**/hdfs2/hadoop/yarn/local/usercache/hdfs/appcache/application_1432831236474_0003/container_1432831236474_0003_01_000002/**

Searching for this directory will prove to be problematic because these transient files are used by YARN to run the Distributed-Shell application and are removed once the application finishes. You can preserve these files for a specific interval by adding the following lines to the yarn-site.xmlconfiguration file and restarting YARN:

```
<property>
   <name>yarn.nodemanager.delete.debug-delay-sec</name>
   <value>100000</value>
</property>
```

Choose a delay, in seconds, to preserve these files, and remember that all applications will create these files. If you are using Ambari, look on the YARN Configs tab under the Advanced yarn-site options, make the change and restart YARN. (See Chapter 9, "Managing Hadoop with Apache Ambari," for more information on Ambari administration.) These files will be retained on the individual nodes only for the duration of the specified delay.

When debugging or investigating YARN applications, these files—in particular, launch_container.sh—offer important information about YARN processes. Distributed-Shell can be used to see what this file contains. Using DistributedShell, the contents of the launch_container.sh file can be printed with the following command:

$ **yarn org.apache.hadoop.yarn.applications.distributedshell.Client -jar $YARN_DS -shell_command cat -shell_args launch_container.sh**

This command prints the launch_container.sh file that is created and run by YARN. The contents of this file are shown in Listing 8.1. The file basically exports some important YARN variables and then, at the end, "execs" the command (cat launch_container.sh) directly and sends any output to logs.

Listing 8.1 **Distributed-Shell launch_container.sh File**

```
#!/bin/bash

export NM_HTTP_PORT="8042"
export LOCAL_DIRS="/opt/hadoop/yarn/local/usercache/hdfs/appcache/
application_1432816241597_0004,/hdfs1/hadoop/yarn/local/usercache/hdfs/appc
ache/
application_1432816241597_0004,/hdfs2/hadoop/yarn/local/usercache/hdfs/appc
ache/
application_1432816241597_0004"
export JAVA_HOME="/usr/lib/jvm/java-1.7.0-openjdk.x86_64"
export
NM_AUX_SERVICE_mapreduce_shuffle="AAA0+gAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAA=
"
export HADOOP_YARN_HOME="/usr/hdp/current/hadoop-yarn-client"
export HADOOP_TOKEN_FILE_LOCATION="/hdfs2/hadoop/yarn/local/usercache/hdfs/
appcache/application_1432816241597_0004/container_1432816241597_0004_01_000
002/
container_tokens"
export NM_HOST="limulus"
export JVM_PID="$$"
export USER="hdfs"
export PWD="/hdfs2/hadoop/yarn/local/usercache/hdfs/appcache/
application_1432816241597_0004/container_1432816241597_0004_01_000002"
export CONTAINER_ID="container_1432816241597_0004_01_000002"
export NM_PORT="45454"
export HOME="/home/"
export LOGNAME="hdfs"
export HADOOP_CONF_DIR="/etc/hadoop/conf"
```

```
export MALLOC_ARENA_MAX="4"
export LOG_DIRS="/opt/hadoop/yarn/log/application_1432816241597_0004/
container_1432816241597_0004_01_000002,/hdfs1/hadoop/yarn/log/
application_1432816241597_0004/container_1432816241597_0004_01_000002,/hdfs
2/
hadoop/yarn/log/application_1432816241597_0004/
container_1432816241597_0004_01_000002"
exec /bin/bash -c "cat launch_container.sh
1>/hdfs2/hadoop/yarn/log/application_1432816241597_0004/
container_1432816241597_0004_01_000002/stdout 2>/hdfs2/hadoop/yarn/log/
application_1432816241597_0004/container_1432816241597_0004_01_000002/stder
r "
hadoop_shell_errorcode=$?
if [ $hadoop_shell_errorcode -ne 0 ]
then
  exit $hadoop_shell_errorcode
fi
```

There are more options for the Distributed-Shell that you can test. The real value of the Distributed-Shell application is its ability to demonstrate how applications are launched within the Hadoop YARN infrastructure. It is also a good starting point when you are creating YARN applications.

## STRUCTURE OF YARN APPLICATIONS

The structure and operation of a YARN application are covered briefly in this section.

The central YARN ResourceManager runs as a scheduling daemon on a dedicated machine and acts as the central authority for allocating resources to the various competing applications in the cluster. The ResourceManager has a central and global view of all cluster resources and, therefore, can ensure fairness, capacity, and locality are shared across all users. Depending on the application demand, scheduling priorities, and resource availability, the ResourceManager dynamically allocates resource containers to applications to run on particular nodes. A container is a logical bundle of resources (e.g., memory, cores) bound to a particular cluster node. To enforce and track such assignments, the ResourceManager interacts with a special system daemon running on each node called the NodeManager. Communications between the ResourceManager and NodeManagers are heartbeat based for scalability. NodeManagers are responsible for local monitoring of resource availability, fault reporting, and container life-cycle management (e.g., starting and killing jobs). The ResourceManager depends on the NodeManagers for its “global view” of the cluster.

User applications are submitted to the ResourceManager via a public protocol and go through an admission control phase during which security credentials are validated and various operational and administrative checks are performed. Those applications that are accepted

pass to the scheduler and are allowed to run. Once the scheduler has enough resources to satisfy the request, the application is moved from an accepted state to a running state. Aside from internal bookkeeping, this process involves allocating a container for the single ApplicationMaster and spawning it on a node in the cluster. Often called container 0, the ApplicationMaster does not have any additional resources at this point, but rather must request additional resources from the ResourceManager.

The ApplicationMaster is the "master" user job that manages all application life-cycle aspects, including dynamically increasing and decreasing resource consumption (i.e., containers), managing the flow of execution (e.g., in case of MapReduce jobs, running reducers against the output of maps), handling faults and computation skew, and performing other local optimizations. The ApplicationMaster is designed to run arbitrary user code that can be written in any programming language, as all communication with the ResourceManager and NodeManager is encoded using extensible network protocols

YARN makes few assumptions about the ApplicationMaster, although in practice it expects most jobs will use a higher-level programming framework. By delegating all these functions to ApplicationMasters, YARN's architecture gains a great deal of scalability, programming model flexibility, and improved user agility. For example, upgrading and testing a new MapReduce framework can be done independently of other running MapReduce frameworks.

Typically, an ApplicationMaster will need to harness the processing power of multiple servers to complete a job. To achieve this, the ApplicationMaster issues resource requests to the ResourceManager. The form of these requests includes specification of locality preferences (e.g., to accommodate HDFS use) and properties of the containers. The ResourceManager will attempt to satisfy the resource requests coming from each application according to availability and scheduling policies. When a resource is scheduled on behalf of an ApplicationMaster, the ResourceManager generates a lease for the resource, which is acquired by a subsequent ApplicationMaster heartbeat. The ApplicationMaster then works with the NodeManagers to start the resource. A token-based security mechanism guarantees its authenticity when the ApplicationMaster presents the container lease to the NodeManager. In a typical situation, running containers will communicate with the ApplicationMaster through an application-specific protocol to report status and health information and to receive framework-specific commands. In this way, YARN provides a basic infrastructure for monitoring and life-cycle management of containers, while each framework manages

application-specific semantics independently. This design stands in sharp contrast to the original Hadoop version 1 design, in which scheduling was designed and integrated around managing only MapReduce tasks.

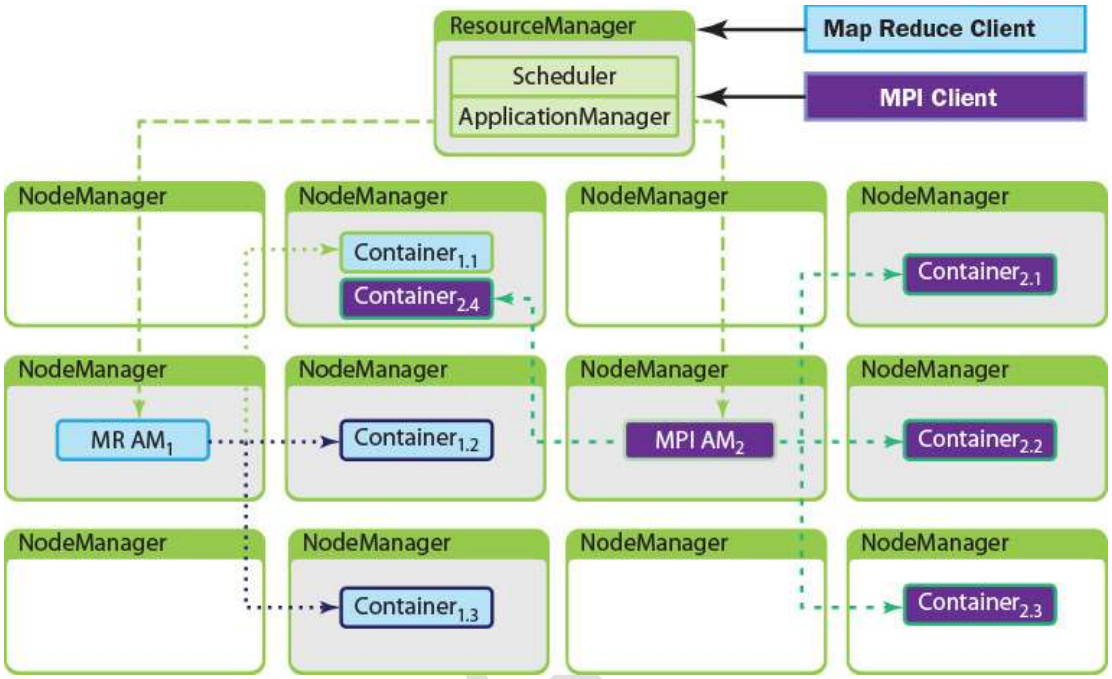



Figure 8.1 YARN architecture with two clients (MapReduce and MPI). The darker client (MPI $AM_2$) is running an MPI application, and the lighter client (MR $AM_1$) is running a MapReduce application. (From Arun C. Murthy, et al., *Apache Hadoop™ YARN*, copyright © 2014, p. 45. Reprinted and electronically reproduced by permission of Pearson Education, Inc., New York, NY.)

## YARN APPLICATION FRAMEWORKS

One of the most exciting aspects of Hadoop version 2 is the capability to run all types of applications on a Hadoop cluster. In Hadoop version 1, the only processing model available to users is MapReduce. In Hadoop version 2, MapReduce is separated from the resource management layer of Hadoop and placed into its own application framework. Indeed, the growing number of YARN applications offers a high level and multifaceted interface to the Hadoop data lake.

YARN presents a resource management platform, which provides services such as scheduling, fault monitoring, data locality, and more to MapReduce and other frameworks. Figure 8.2 illustrates some of the various frameworks that will run under YARN. Note that the Hadoop version 1 applications (e.g., Pig and Hive) run under the MapReduce framework.

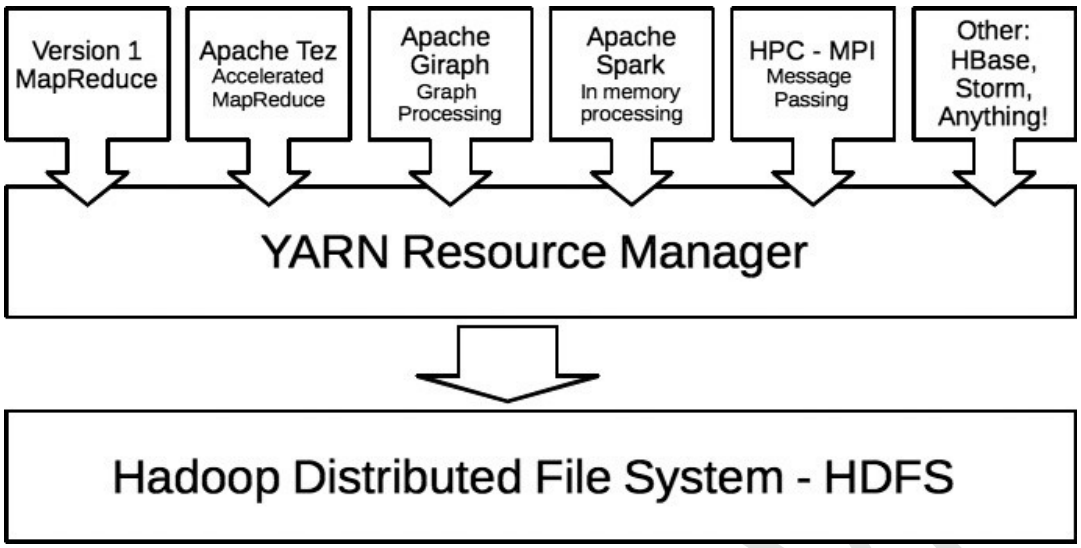



Figure 8.2 Example of the Hadoop version 2 ecosystem. Hadoop version 1 supports batch MapReduce applications only.

This section presents a brief survey of emerging open source YARN application frameworks that are being developed to run under YARN. As of this writing, many YARN frameworks are under active development and the framework landscape is expected to change rapidly. Commercial vendors are also taking advantage of the YARN platform. Consult the webpage for each individual framework for full details of its current stage of development and deployment.

**Distributed-Shell**

As described earlier in this chapter, Distributed-Shell is an example application included with the Hadoop core components that demonstrates how to write applications on top of YARN. It provides a simple method for running shell commands and scripts in containers in parallel on a Hadoop YARN cluster.

**Hadoop MapReduce**

MapReduce was the first YARN framework and drove many of YARN's requirements. It is integrated tightly with the rest of the Hadoop ecosystem projects, such as Apache Pig, Apache Hive, and Apache Oozie.

**Apache Tez**

One great example of a new YARN framework is Apache Tez. Many Hadoop jobs involve the execution of a complex directed acyclic graph (DAG) of tasks using separate MapReduce

stages. Apache Tez generalizes this process and enables these tasks to be spread across stages so that they can be run as a single, all-encompassing job.

Tez can be used as a MapReduce replacement for projects such as Apache Hive and Apache Pig. No changes are needed to the Hive or Pig applications.

**Apache Giraph**

Apache Giraph is an iterative graph processing system built for high scalability. Facebook, Twitter, and LinkedIn use it to create social graphs of users. Giraph was originally written to run on standard Hadoop V1 using the MapReduce framework, but that approach proved inefficient and totally unnatural for various reasons. The native Giraph implementation under YARN provides the user with an iterative processing model that is not directly available with MapReduce. Support for YARN has been present in Giraph since its own version 1.0 release. In addition, using the flexibility of YARN, the Giraph developers plan on implementing their own web interface to monitor job progress

**Hoya: HBase on YARN**

The Hoya project creates dynamic and elastic Apache HBase clusters on top of YARN. A client application creates the persistent configuration files, sets up the HBase cluster XML files, and then asks YARN to create an ApplicationMaster. YARN copies all files listed in the client's application-launch request from HDFS into the local file system of the chosen server, and then executes the command to start the Hoya ApplicationMaster. Hoya also asks YARN for the number of containers matching the number of HBase region servers it needs.

**Dryad on YARN**

Similar to Apache Tez, Microsoft's Dryad provides a DAG as the abstraction of execution flow. This framework is ported to run natively on YARN and is fully compatible with its non-YARN version. The code is written completely in native C++ and C# for worker nodes and uses a thin layer of Java within the application.

**Apache Spark**

Spark was initially developed for applications in which keeping data in memory improves performance, such as iterative algorithms, which are common in machine learning, and interactive data mining. Spark differs from classic MapReduce in two important ways. First, Spark holds intermediate results in memory, rather than writing them to disk. Second, Spark

supports more than just MapReduce functions; that is, it greatly expands the set of possible analyses that can be executed over HDFS data stores. It also provides APIs in Scala, Java, and Python.

Since 2013, Spark has been running on production YARN clusters at Yahoo!. The advantage of porting and running Spark on top of YARN is the common resource management and a single underlying file system.

**Apache Storm**

Traditional MapReduce jobs are expected to eventually finish, but Apache Storm continuously processes messages until it is stopped. This framework is designed to process unbounded streams of data in real time. It can be used in any programming language. The basic Storm use-cases include real-time analytics, online machine learning, continuous computation, distributed RPC (remote procedure calls), ETL (extract, transform, and load), and more. Storm provides fast performance, is scalable, is fault tolerant, and provides processing guarantees. It works directly under YARN and takes advantage of the common data and resource management substrate.

**Apache REEF: Retainable Evaluator Execution Framework**

YARN's flexibility sometimes requires significant effort on the part of application implementers. The steps involved in writing a custom application on YARN include building your own ApplicationMaster, performing client and container management, and handling aspects of fault tolerance, execution flow, coordination, and other concerns. The REEF project by Microsoft recognizes this challenge and factors out several components that are common to many applications, such as storage management, data caching, fault detection, and checkpoints. Framework designers can build their applications on top of REEF more easily than they can build those same applications directly on YARN, and can reuse these common services/libraries. REEF's design makes it suitable for both MapReduce and DAG-like executions as well as iterative and interactive computations.

**Hamster: Hadoop and MPI on the Same Cluster**

The Message Passing Interface (MPI) is widely used in high-performance computing (HPC). MPI is primarily a set of optimized message-passing library calls for C, C++, and Fortran that operate over popular server interconnects such as Ethernet and InfiniBand. Because users have full control over their YARN containers, there is no reason why MPI applications cannot run within a Hadoop cluster. The Hamster effort is a work-in-progress that provides a good discussion of the issues involved in mapping MPI to a YARN cluster.

### Apache Flink: Scalable Batch and Stream Data Processing

Apache Flink is a platform for efficient, distributed, general-purpose data processing. It features powerful programming abstractions in Java and Scala, a high-performance run time, and automatic program optimization. It also offers native support for iterations, incremental iterations, and programs consisting of large DAGs of operations.

Flink is primarily a stream-processing framework that can look like a batch-processing environment. The immediate benefit from this approach is the ability to use the same algorithms for both streaming and batch modes (exactly as is done in Apache Spark). However, Flink can provide low-latency similar to that found in Apache Storm, but which is not available in Apache Spark.

In addition, Flink has its own memory management system, separate from Java's garbage collector. By managing memory explicitly, Flink almost eliminates the memory spikes often seen on Spark clusters.

### Apache Slider: Dynamic Application Management

Apache Slider (incubating) is a YARN application to deploy existing distributed applications on YARN, monitor them, and make them larger or smaller as desired in real time.

Applications can be stopped and then started; the distribution of the deployed application across the YARN cluster is persistent and allows for best-effort placement close to the previous locations. Applications that remember the previous placement of data (such as HBase) can exhibit fast startup times by capitalizing on this feature.

YARN monitors the health of "YARN containers" that are hosting parts of the deployed applications. If a container fails, the Slider manager is notified. Slider then requests a new replacement container from the YARN ResourceManager. Some of Slider's other features include user creation of on-demand applications, the ability to stop and restart applications as needed (preemption), and the ability to expand or reduce the number of application containers as needed. The Slider tool is a Java command-line application.

# Module -3

# NoSQL

## 3.1 Introduction

Big Data uses distributed systems. A distributed system consists of multiple data nodes at clusters of machines and distributed software components. The tasks execute in parallel with data at nodes in clusters. The computing nodes communicate with the applications through a network.

Following are the features of distributed-computing architecture (Chapter

l. **Increased reliability and fault tolerance:** The important advantage of distributed computing system is reliability. If a segment of machines in a cluster fails then the rest of the machines continue work. When the datasets replicate at number of data nodes, the fault tolerance increases further. The dataset in remaining segments continue the same computations as being done at failed segment machines.

**2. Flexibility** makes it very easy to install, implement and debug new services in a distributed environment.

**3. Sharding** is storing the different parts of data onto different sets of data nodes, clusters or servers. For example, university students huge database, on sharding divides in databases, called shards. Each shard may correspond to a database for an individual course and year. Each shard stores at different nodes or servers.

**4. Speed:** Computing power increases in a distributed computing system as shards run parallelly on individual data nodes in clusters independently (no data sharing between shards).

**5. Scalability:** Consider sharding of a large database into a number of shards, distributed for computing in different systems. When the database expands further, then adding more machines and increasing the number of shards provides horizontal scalability. Increased computing power and running number of algorithms on the same machines provides vertical scalability.Resources sharing: Shared resources of memory, machines and network architecture reduce the cost.

Open system makes the service accessible to all nodes.

**6. Performance:** The collection of processors in the system provides higher performance than a centralized computer, due to lesser cost of communication among machines (Cost means time taken up in communication).

## 3.2 NOSQL DATA STORE

SQL is a programming language based on relational algebra. It is a declarative language and it defines the data schema . SQL creates databases and RDBMS s. RDBMS uses tabular data store with relational algebra, precisely defined operators with relations as the operands. Relations are a set of tuples. Tuples are named attributes. A tuple identifies uniquely by keys called candidate keys.

### *ACID Properties in SQL Transactions*

**Atomicity** of transaction means all operations in the transaction must complete, and if interrupted, then must be undone (rolled back). For example, if a customer withdraws an amount then the bank in first operation enters the withdrawn amount in the table and in the next operation modifies the balance with new amount available. Atomicity means both should be completed, else undone if interrupted in between.

**Consistency** in transactions means that a transaction must maintain the integrity constraint, and follow the consistency principle. For example, the difference of sum of deposited amounts and withdrawn amounts in a bank account must equal the last balance. All three data need to be consistent.

**Isolation** of transactions means two transactions of the database must be isolated from each other and done separately.

**Durability** means a transaction must persist once completed

**NOSQL**

A new category of data stores is NoSQL (means Not Only SQL) data stores. NoSQL is an altogether new approach of thinking about databases, such as schema flexibility, simple relationships, dynamic schemas, auto sharding, replication, integrated caching, horizontal scalability of shards, distributable tuples, semi-structures data and flexibility in approach.

Issues with NoSQL data stores are lack of standardization in approaches, processing difficulties for complex queries, dependence on eventually consistent results in place of consistency in all states.

### *Big Data NoSQL*

NoSQL records are in non-relational data store systems. They use flexible data models. The records use multiple schemas. NoSQL data stores are considered as semi-structured data. Big Data Store uses NoSQL.

**NoSQL data store characteristics are as follows:**

1. NoSQL is a class of non-relational data storage system with flexible data model. Examples of NoSQL data-architecture patterns of datasets are key-value pairs, name/value pairs, Column family,Big-data store, Tabular data store, Cassandra (used in Facebook/Apache), HBase, hash table [Dynamo (Amazon S3)], unordered keys using ]SON (CouchDB), ]SON (PNUTS), ]SON (MongoDB), Graph Store, Object Store, ordered keys and semi-structured data storage systems.
2. NoSQL not necessarily has a fixed schema, such as table; do not use the concept of Joins (in distributed data storage systems); Data written at one node can be replicated to multiple nodes. Data store is thus fault- tolerant. The store can be partitioned into unshared shards.

**Features in NoSQL Transactions** NoSQL transactions have following features:

1. **Relax one or more of the ACID properties.**
2. Characterize by two out of three properties (consistency, availability and partitions) of CAP theorem, two are at least present for the application/ service/process.
3. Can be characterized by BASE properties

**Big Data NoSQL Solutions** NoSQL DBs are needed for Big Data solutions. They play an important role in handling Big Data challenges. Table 3.1 gives the examples of widely used NoSQL data stores.

**Table 3.1** NoSQL data stores and their characteristic features

| | |
|---|---|
| Apache's HBase | HDFS compatible, open-source and non-relational data store written inJava; A column-family based NoSQL data store, data store providing BigTable-like capabilities (Sections 2.6 and 3.3.3.2); scalability, strong consistency, versioning, configuring and maintaining data store characteristics |
| Apache's MongoDB | HDFS compatible; master-slave distribution model (Section 3.5.1.3); document-oriented data store withJSON-like documents and dynamic schemas; open-source, NoSQL, scalable and non-relational database; used by Websites Craigslist, eBay, Foursquare at the backend |
| Apache's Cassandra | HDFS compatible DBs; decentralized distribution peer-to-peer model (Section 3.5.1.4); open source; NoSQL; scalable, non-relational, column-family based, fault-tolerant and tuneable consistency (Section 3.7) used by Facebook and Instagram |

| | |
|---|---|
| Apache's CouchDB | A project of Apache which is also widely used database for the web. CouchDB consists of Document Store. It uses theJSON data exchange format to store its documents,JavaScript for indexing, combining and transforming documents, and HTTP APis |
| Oracle NoSQL | Step towards NoSQL data store; distributed key-value data store; provides transactional semantics for data manipulation , horizontal scalability, simple administration and monitoring |
| Riak | An open-source key-value store; high availability (using replication concept), fault tolerance, operational simplicity, scalability and written in Erlang |

**CAP Theorem** Among C, A and P, two are at least present for the application/service/process. Consistency means all copies have the same value like in traditional DBs. Availability means at least one copy is available in case a partition becomes inactive or fails. For example, in web applications, the other copy in the other partition is available. Partition means parts which are active but may not cooperate (share) as in distributed DBs.

1. *Consistency in distributed databases* means that all nodes observe the same data at the same time. Therefore, the operations in one partition of the database should reflect in other related partitions in case of distributed database. Operations, which change the sales data from a specific showroom in a table should also reflect in changes in related tables which are using that sales data.
2. *Availability* means that during the transactions, the field values must be available in other partitions of the database so that each request receivesa response on success as well as failure. (Failure causes the response to request from the replicate of data). Distributed databases require transparency between one another. Network failure may lead to data unavailability in a certain partition in case of no replication. Replication ensures availability.
3. *Partition* means division of a large database into different databases without affecting the operations on them by adopting specified procedures.
4. *Partition tolerance:* Refers to continuation of operations as a whole even in case of message loss, node failure or node not reachable.

Brewer's CAP (c.onsistency, Availability and fartition Tolerance) theorem demonstrates that any distributed system cannot guarantee C, A and P together.

1. Consistency- All nodes observe the same data at the same time.
2. Availability- Each request receives a response on success/failure.
3. Partition Tolerance-The system continues to operate as a whole even in case of message loss, node failure or node not reachable.

Partition tolerance cannot be overlooked for achieving reliability in a distributed database system. Thus, in case of any network failure, a choice canbe:

- Database must answer, and that answer would be old or wrong data (AP).
- Database should not answer, unless it receives the latest copy of the data(CP).

The CAP theorem implies that for a network partition system, the choice of consistency and availability are mutually exclusive. CA means consistency andavailability, AP means availability and partition tolerance and CP means consistency and partition tolerance. Figure 3.1 shows the CAP theorem usage in Big Data Solutions.

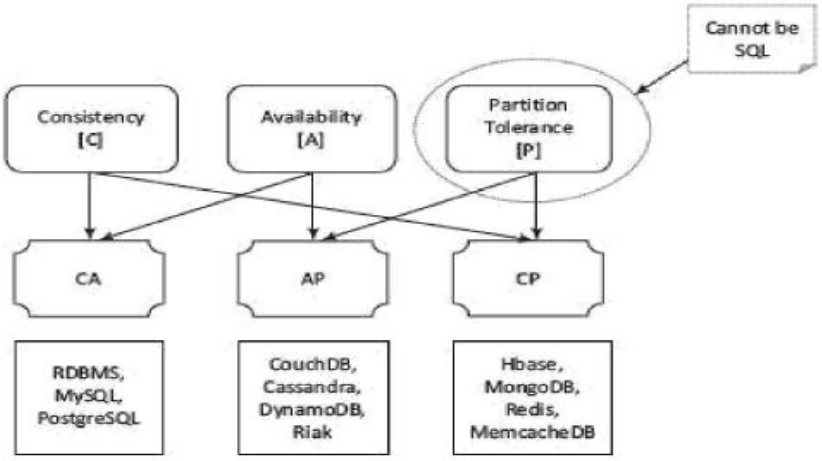



**Figure 3.1** CAP theorem in Big Data solutions

**Schema Less Database**

Schema of a database system refers to designing of a structure for datasets and data structures for storing into the database. NoSQL data not necessarily have a fixed table schema. The systems do not use the concept of Join (between distributed datasets). A cluster-based highly distributed node manages a single large data store with a NoSQL DB. Data written at one node replicates to multiple nodes. Therefore, these are identical, fault-tolerant and partitioned into shards. Distributed databases can store and process a set of information on more than one computing nodes.

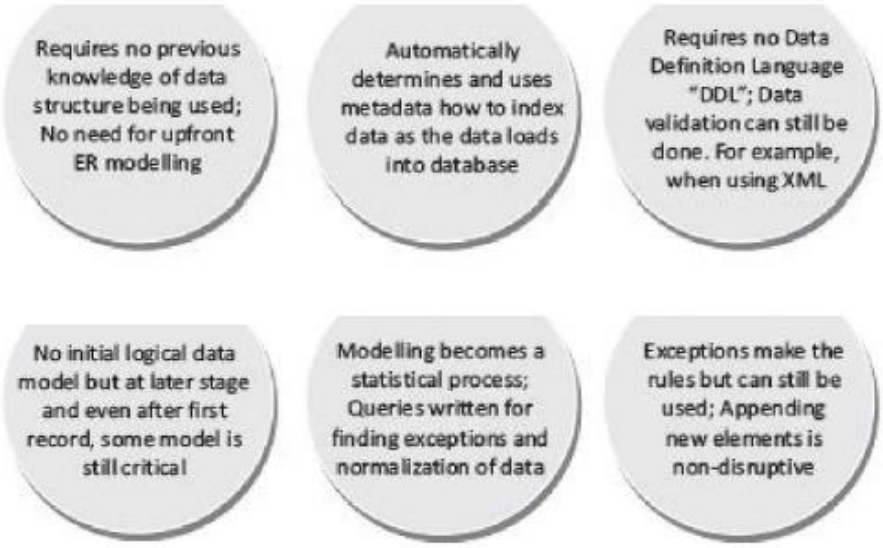



**Figure 3.2** Characteristics of Schema-less model

Increasing Flexibility for Data Manipulation

NoSQL data store possess characteristic of increasing flexibility for data manipulation. The new attributes to database can be increasingly added. Late binding of them is also permitted.

BASE Properties BA stands for basic availability, S stands for soft state and E stands for eventual consistency.

l. Basic availability ensures by distribution of shards (many partitions of huge data store) across many data nodes with a high degree of replication. Then, a segment failure does not necessarily mean a complete data store unavailability.

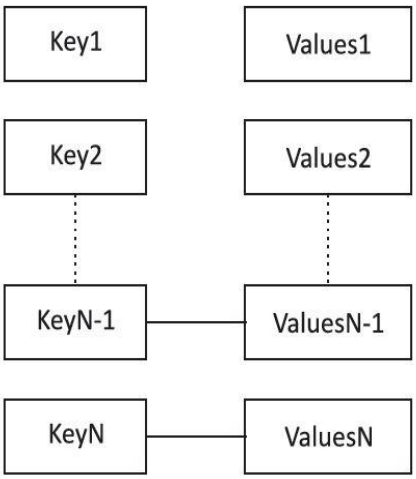
2. Soft state ensures processing even in the presence of inconsistencies but achieving consistency eventually. A program suitably takes into account the inconsistency found during processing. NoSQL database design does not consider the need of consistency all along the processing time.

3. Eventual consistency means consistency requirement in NoSQL databases meeting at some point of time in future. Data converges eventually to a consistent state with no time-frame specification for achieving that. ACID rules require consistency all along the processing on completion of each transaction. BASE does not have that requirement and has the flexibility.

## 3.3 NOSQL DATA ARCHITECTURE PATTERNS

### 3.3.1 Key-Value Store

The simplest way to implement a schema-less data store is to use key-value pairs. The data store characteristics are high performance, scalability and flexibility. Data retrieval is fast in key-value pairs data store. A simple string called, key maps to a large data string or BLOB (Basic Large Object). Key-value store accesses use a primary key for accessing the values. Therefore, the store can be easily scaled up for very large data. The concept is similar to a hash table where a unique key points to a particular item(s) of data. Figure 3.4 shows key-value pairs architectural pattern and example of students' database as key-value pairs




| Key | Value |
|---|---|
| "Ashish" | "Category: Student; Class: B.Tech.; Semester: VII; Branch: Engineering; Mobile:9999912345" |
| "Mayuri" | "Category: student; class: M.Tech.; Mobile:8888823456" |

**Advantages of a key-value store are as follows:**

1. Data Store can store any data type in a value field. The key-value system stores the information as a BLOB of data (such as text, hypertext, images,video and audio) and return the same BLOB when the data is retrieved. Storage is like an English dictionary. Query for a word retrieves the meanings, usages, different forms as a single item in the dictionary. Similarly, querying for key retrieves the values.
2. A query just requests the values and returns the values as a single item. Values can be of any data type.
3. Key-value store is eventually consistent.
4. Key-value data store may be hierarchical or may be ordered key-value store.
5. Returned values on queries can be used to convert into lists, table- columns, data-

frame fields and columns.

6. Have (i) scalability, (ii) reliability, (iii) portability and (iv) low operationalcost.
7. The key can be synthetic or auto-generated. The key is flexible and can be represented in many formats: (i) Artificially generated strings created from a hash of a value, (ii) Logical path names to images or files, (iii) RESTweb-service calls (request response cycles), and (iv) SQL queries.

**Limitations of key-value store architectural pattern are:**

1. No indexes are maintained on values, thus a subset of values is not searchable.
2. Key-value store does not provide traditional database capabilities, such as atomicity of transactions, or consistency when multiple transactions are executed simultaneously. The application needs to implement such capabilities.
3. Maintaining unique values as keys may become more difficult when the volume of data increases. One cannot retrieve a single result when a key- value pair is not uniquely identified.
4. Queries cannot be performed on individual values. No clause like 'where' in a relational database usable that filters a result set.

**Table 3.2** Traditional relational data model vs. the key-value store model

| Traditional relational model | Key-value store model |
|---|---|
| Result set based on row values | Queries return a single item |
| Values of rows for large datasets are indexed | No indexes on values |
| Same data type values in columns | Any data type values |

Typical uses of key-value store are:

(i) Image store,

(ii) Document or file store,

(iii) Lookup table, and

(iv) Query-cache.

***Riak*** is open-source Erlang language data store. It is a key-value data store system. Data auto-

distributes and replicates in Riak. It is thus, fault tolerant and reliable. Some other widely used key-value pairs in NoSQL DBs are Amazon's DynamoDB, Redis (often referred as Data Structure server), Memcached and its flavours, Berkeley DB, upscaledb (used for embedded databases), project Voldemort and Couchbase.

### 3.3.2 Document Store

Characteristics of Document Data Store are high performance and flexibility. Scalability varies, depends on stored contents. Complexity is low compared to tabular, object and graph data stores.

Following are the features in Document Store:

1. Document stores unstructured data.
2. Storage has similarity with object store.
3. Data stores in nested hierarchies. For example, inJSON formats data model[Example 3.3(ii)], XML document object model (DOM), or machine-readable data as one BLOB. Hierarchical information stores in a single unit called *document tree*. Logical data stores together in a unit.
4. Querying is easy. For example, using section number, sub-section number and figure caption and table headings to retrieve document partitions.
5. No object relational mapping enables easy search by following paths fromthe root of document tree.
6. Transactions on the document store exhibit ACID properties.

Typical uses of a document store are: (i) office documents, (ii) inventory store, (iii) forms data, (iv) document exchange and (v) document search.

Examples of Document Data Stores are CouchDB and MongoDB.

**CSV and JSON File Formats** CSV data store is a format for records CSV does not represent object-oriented databases or hierarchical data records. *JSON* and XML represent semistructured data, object- oriented records and hierarchical data records. *JSON* (Java Script Object Notation) refers to a language format for semistructured data. *JSON* represents object-oriented and hierarchical data records, object, and resource arrays in JavaScript.

Semester, Subject Code, Subject Name, Grade
1, CS101, ""Theory of Computations"", 7.8.
1, CS102,1, ""Computer Architecture"", 7.8.
.....
2, CS204, ""Object Oriented Programming"", 7.2.
2, CS205, ""Data Analytics"", 8.1.

**JSON Files**

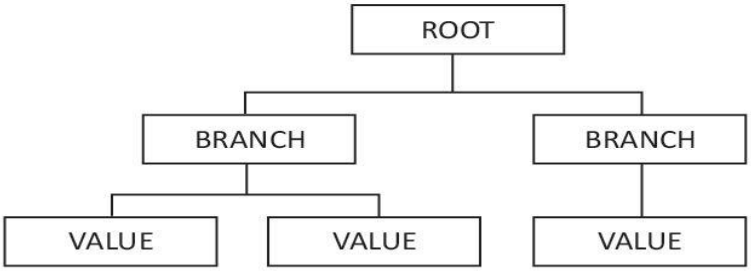
- Semi-structured data
- object-oriented records and hierarchical data records
- JSON refers to a language format for semistructured data. JSON represents object-oriented and hierarchical data records, object, and resource arrays in JavaScript

**Document JSON Format CouchDB Database** Apache CouchDB is an open- source database. Its features are:

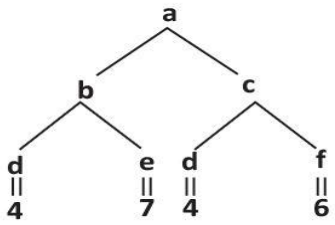
- CouchDB provides mapping functions during querying, combining and filtering of information.
- CouchDB deploys JSON Data Store model for documents. Each document maintains separate data and metadata (schema).
- CouchDB is a multi-master application. Write does not require field locking when controlling the concurrency during multi-master application.
- CouchDB querying language is JavaScript. Java script is a language which

## XML

- An extensible, simple and scalable language. Its self-describing format describes structure and contents in an easy to understand format
- XML is widely used. The document model consists of root element and their sub-elements. XML document model has a hierarchical structure. XML document model has features of object-oriented records. XML format finds wide uses in data store and
- XML document model has a hierarchical structure. XML document model has features of object-oriented records. XML format finds wide uses in data store




```
<a>
   <b>
        <d> 4 </d>
        <e> 7 </e>
   </b>
   <c>
        <d> 4 </d>
        <f> 6 </f>
   </c>
</a>
```




(a) XML document fragment (b) Tree representation of fragment

**Tabular data** stores use rows and columns. Row-head field may be used as a keywhich access and retrieves multiple values from the successive columns in that row. The OLTP is fast on in-memory row-format data.

**Columnar Data Store** A way to implement a schema is the divisions into columns. Storage of each column, successive values is at the successive memory addresses. Analytics processing (AP) In-memory uses columnar storage in memory. A pair of row-head and column-head is a key-pair. The pair accesses a field in the table.

**Column-Family Data Store** Column-family data-store has a group of columns as a column family. A combination of row-head, column-family head and table- column head can also be a key to access a field in a column of the table during querying. Combination of row head, column families head, column-family head and column head for values in column fields can

also be a key to access fields ofa column. A column-family head is also called a super-column head.

**Sparse Column Fields** A row may associate a large number of columns but contains values in few column fields. Similarly, many column fields may not have data. Columns are logically grouped into column families. Column-family data stores are then similar to sparse matrix data. Most elements of sparse matrix are empty. Data stores at memory addresses is columnar-family based rather than as row based. Metadata provide the column-family indices of not empty column fields.

That facilitates OLAP of not empty column families faster. For example, assume hash key in a column heading field and values in successive rows at one column family. For another key, the values will be in another column family.

**Grouping of Column Families** Two or more column-families in data store form a super group, called super column. Table 3.3 consists of one such group (super column), 'Nestle Chocolate Flavours Group'.

Grouping into Rows When number of rows are very large then horizontal partitioning of the table is a necessity. Each partition forms one row-group. For example, a group of 1 million rows per partition. A row group thus has all column data store in the memory for in-memory analytics. Practically, row groups are chosen such that memory required for the group is above, say 10 MB and below the maximum size which can cached and buffered in memory, say 1 GB for in-memory analytics.

**Characteristics of Columnar Family Data Store** Columnar family data store imbibes characteristics of very high performance and scalability, moderate level

of flexibility and lower complexity when compared to the object and graph databases. Advantages of column stores are:

l. *Scalability:* The database uses row IDs and column names to locate a column and values at the column fields. The interface for the fields is simple. The back-end system can distribute queries over a large number of processing nodes without performing any Join operations. The retrieval of data from the distributed node can be least complicated by an intelligent plan of row IDs and columns, thereby increasing performance. Scalability means addition of number of rows as the number of ACVMs increase in Example1.6(i). Number of processing instructions is proportional to the number of ACVMs due to scalable operations.

2. *Partitionability:* For example, large data of ACVMs can be partitioned into datasets of size, say 1 MB in the number of row-groups. Values in columns of each row-group,process in-memory at a partition. Values in columns of each row-group independently parallelly process in-memory at the partitioned nodes.
3. *Availability:* The cost of replication is lower since the system scales on distributed nodes efficiently. The lack of Join operations enables storing a part of a column- family matrix on remote computers. Thus, the data is always available in case of failure of any node.
4. *Tree-like columnar* structure consisting of column-family groups, column families and columns. The columns group into families. The column families group into column groups (super columns). A key for the column fields consists of three secondary keys: column-families group ID, column-family ID and column-head name.
5. *Adding new data at* ease: Permits new column *Insert* operations. Trigger operation creates new columns on an Insert. The column-field values can add after the last address in memory if the column structure is known in advance. New row-head field, row-group ID field, column-family group, column family and column names can be created at any time to add new data.
6. Querying all the field values in a column in a family, all columns in the family or a group of column-families, is fast in in-memory column-family data store.
7. Replication of columns: HDFS-compatible column-family data stores replicate each data store with default replication factor= 3.
8. No optimization for Join: Column-family data stores are similar to sparse matrix data. The data do not optimize for Join operations.

**Big Table Data Store**

Examples of widely used column-family data store are Google's BigTable, HBase and Cassandra. Keys for *row key, column key, timestamp* and *attribute* uniquely identify the values in the fields

Following are features of a BigTable:

- Massively scalable NoSQL. BigTable scales up to 100s of petabytes.
- Integrates easily with Hadoop and Hadoop compatible systems.
- Compatibility with MapReduce, HBase APis which are open-source Big Data platforms.
- Key for a field uses not only row_ID and Column_ID (for example, ACVM_ID and KitKat

in Example 3.6) but also timestamp and attributes. Values are ordered bytes. Therefore, multiple versions of values may be present in the BigTable.

- Handles million of operations per second.
- Handle large workloads with low latency and high throughput
- Consistent low latency and high throughput
- APIs include security and permissions
- BigTable, being Google's cloud service, has global availability and its service is seamless.

### RC *File Format*

Hive uses Record Columnar (RC) file-format records for querying. RC is the best choice for intermediate tables for fast column-family store in HDFS with Hive.Serializability of RC table column data is the advantage. RC file is DeSerializable into column data.

### ORC File Format

An ORC (Optimized Row Columnar) file consists of row-group data called stripes. ORC enables concurrent reads of the same file using separate RecordReaders. Metadata store uses Protocol Buffers for addition and removal of fields. [1]

ORC is an intelligent Big Data file format for HDFS and Hive.[2] An ORC file stores a collections of rows as a row-group. Each row-group data store in columnar format. This enables parallel processing of multiple row-groups in anHDFS cluster.

An ORC file consists of a stripe the size of the file is by default 256 MB. Stripe consists of indexing (mapping) data in 8 columns, row-group columns data (contents) and stripe footer (metadata). An ORC has two sets of columns data instead of one column data in RC. One column is for each map or list size and other values which enable a query to decide skipping or reading of the mapped columns. A mapped column has contents required by the query. The columnar layout in each ORC file thus, optimizes for compression and enables skipping of data in columns. This reduces read and decompression load.

| | | | | | |
|---|---|---|---|---|---|
| Stripe_ID | Index Column 1 | | | | Index Column 2 |
| | Index column 1 key 1 | | | | Index column 2 key 1 |
| | Contents-Column name | Contents Minimum value | Contents Maximum value | Count (number) of content-column fields | |
| | ... | ... | ... | ... | |
| | ... | ... | ... | ... | |
| | Index column 1 key 2 | | | | Index column 2 key 2 |
| | Column-name | Minimum value | Maximum value | Count of number of column fields | |
| | ... | ... | ... | ... | |
| | ... | ... | ... | ... | |

**Keys to access or skip a content column in ORC file format**

### *Parquet File Formats*

Parquet is nested hierarchical columnar-storage concept. Nesting sequence is the table, row group, column chunk and chunk page. Apache Parquet file is columnar-family store file. Apache Spark SQL executes user defined functions (UDFs) which query the Parquet file columns. A programmer writes the codes for an UDF and creates the processing function for big long queries.

A Parquet file uses an HDFS block. The block stores the file for processing queries on Big Data. The file compulsorily consists of metadata, though the file need not consist of data.

The Parquet file consists of row groups. A row-group columns data process in memory after data cache and buffer at the memory from the disk. Each row group has a number of columns. A row group has Ncol columns, and row group consists of Ncol column chunks. This means each column chunk consists of values saved in each column of each row group.

A column chunk can be divided into pages and thus, consists of one or more pages. The column chunk consists of a number of interleaved pages, Npg• A page is a conceptualized unit which can be compressed or encoded together at an instance. The unit is minimum portion of a chunk which is read at an instance for in-memory analytics.

| Row-group_ID | Column Chunk 1 key | | | |
|---|---|---|---|---|
| | Page 1 key | Page 2 key | “” | Page m key |
| | .. | ... | ... | ... |
| | .. | ... | ... | ... |
| | Column Chunk 2 key | | | |
| | Page 1key | Page 2 key | “” | Page key m’ |
| | .. | ... | ... | ... |
| | .. | ... | ... | ... |

Combination of keys for content page in the Parquet file format

**Object Data Store**

An object store refers to a repository which stores the:

1. Objects (such as files, images, documents, folders, and business reports)
2. System metadata which provides information such as filename, creation_date, last_modified, language_used (such as Java, C, C#, C++, Smalltalk, Python), access_permissions, supported query languages)
3. Custom metadata which provides information, such as subject, category, sharing permissions.

Metadata enables the gathering of metrics of objects, searches, finds the contents and specifies the objects in an object data-store tree. Metadata finds the relationships among the objects, maps the object relations and trends. Object Store metadata interfaces with the Big Data. API first mines the metadatato enable mining of the trends and analytics. The metadata defines classes and properties of the objects. Each Object Store may consist of a database. Document content can be stored in either the object store database storage area or in a file storage area. A single file domain may contain multiple Object Stores.

### *Object Relational Mapping*

The following example explains object relational mapping

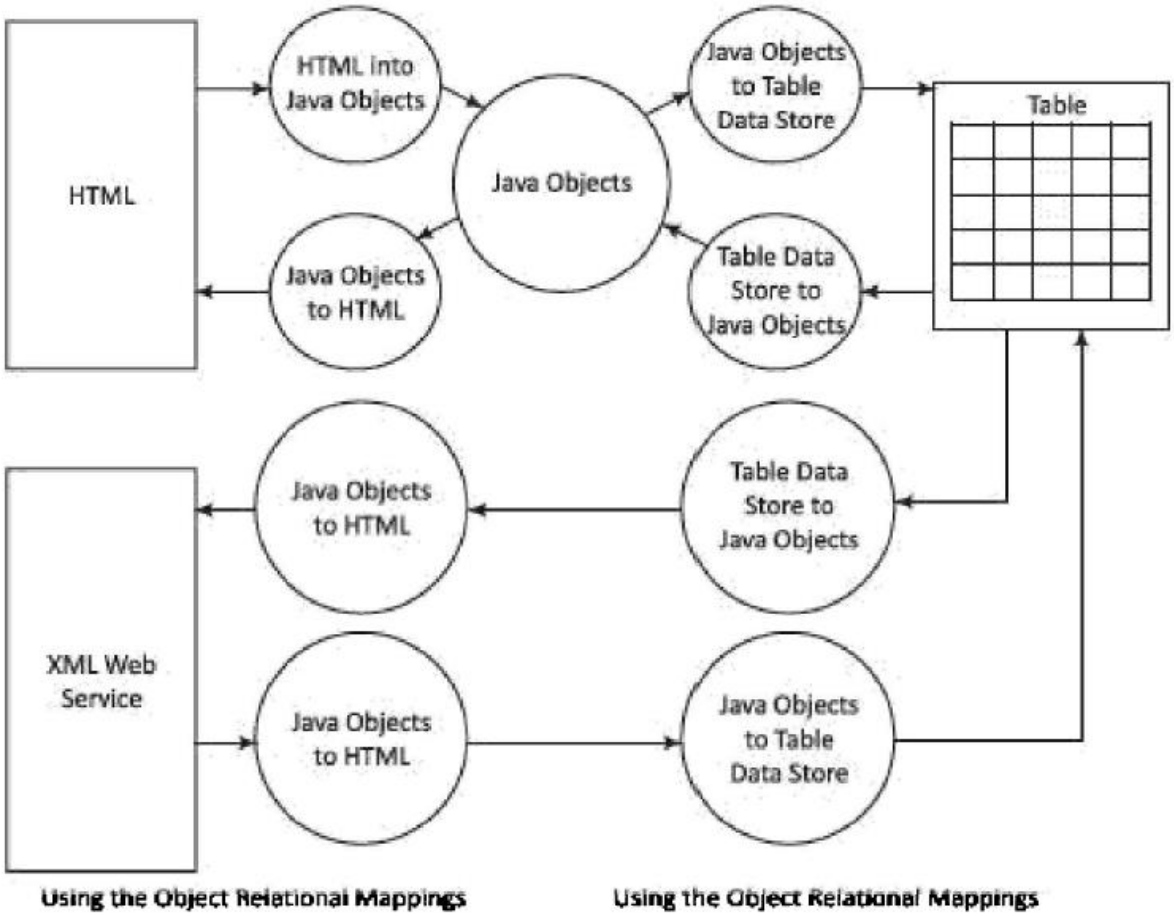
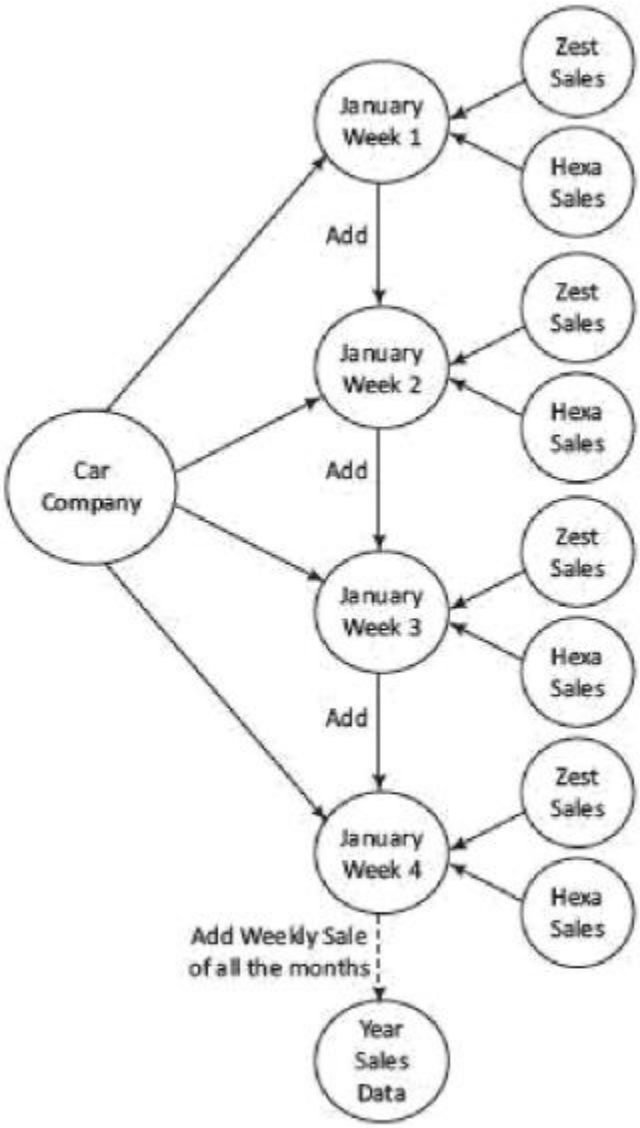



Fig : HTML document and XML web services

## Graph Data Base

One way to implement a data store is to use graph database. A characteristic of graph is high flexibility. Any number of nodes and any number of edges can be added to expand a graph. The complexity is high and the performance is variable with scalability. Data store as series of interconnected nodes. Graph with data nodes interconnected provides one of the best database system when relationships and relationship types have critical values.

Nodes represent entities or objects. Edges encode relationships between nodes. Some operations become simpler to perform using graph models. Examples of graph model usages are social networks of connected people. The connections to related persons become easier to model when using the graph model.




**Graph Data base for Car Model Sale**

Characteristics of graph databases are:

1. Use specialized query languages, such as RDF uses SPARQL
2. Create a database system which models the data in a completely different way than the key-values, document, columnar and object data store models.
3. Can have hyper-edges. A hyper-edge is a set of vertices of a hypergraph. Ahypergraph is a generalization of a graph in which an edge can join any number of vertices (not only the neighbouring vertices).
4. Consists of a collection of small data size records, which have complex interactions between graph-nodes and hypergraph nodes. Nodes represent the entities or objects. Nodes use Joins. Node identification can use URI or other tree-based structure. The edge

encodes a relationship between the nodes.

Graph databases have poor scalability. They are difficult to scale out on multiple servers. This is due to the close connectivity feature of each node in thegraph. Data can be replicated on multiple servers to enhance read and the query processing performance. Write operations to multiple servers and graph queries that span multiple nodes, can be complex to implement.

Typical uses of graph databases are: (i) link analysis, (ii) friend of friend queries, (iii) Rules and inference, (iv) rule induction and (v) Pattern matching. Link analysis is needed to perform searches and look for patterns and relationships in situations, such as social networking, telephone, or email

Examples of graph DBs are Neo4J, AllegroGraph, HyperGraph, Infinite Graph, Titan and FlockDB. Neo4J graph database enable easy usages by Java developers. Neo4J can be designed fully ACID rules compliant. Design consists of adding additional path traversal in between the transactions such that data consistency is maintained and the transactions exhibit ACID properties.

## 3.4 NO SQL to Manage Big Data

**NoSQL Solutions for Big Data**

Big Data solution needs scalable storage of terabytes and petabytes, dropping ofsupport for database Joins, and storing data differently on several distributed servers (data nodes) together as a cluster. A solution, such as CouchDB, DynamoDB, MongoDB or Cassandra follow CAP theorem (with compromising the consistency factor) to make transactions faster and easier to scale. A solution must also be partitioning tolerant

Characteristics of Big Data NoSQL solution are:

l. *High and easy scalability:* NoSQL data stores are designed to expand horizontally. Horizontal scaling means that scaling out by adding more machines as data nodes (servers) into the pool of resources (processing, memory, network connections). The design scales out using multi-utility cloud services.

2. *Support to replication:* Multiple copies of data store across multiple nodes ofa cluster. This ensures high availability, partition, reliability and fault tolerance.

3. Distributable: Big Data solutions permit sharding and distributing of shards on multiple clusters which enhances performance and throughput.

4. *Usages ofNoSQL servers* which are less expensive. NoSQL data stores requireless management efforts. It supports many features like automatic repair, easier data distribution and simpler

data models that makes database administrator (OBA) and tuning requirements less stringent.

5. Usages of open-source tools: NoSQL data stores are cheap and open source. Database implementation is easy and typically uses cheap servers to manage the exploding data and transaction while RDBMS databases are expensive and use big servers and storage systems. So, cost per gigabyte data store and processing of that data can be many times less than the cost ofRDBMS
6. *Support to schema-less data model:* NoSQL data store is schema less, so data can be inserted in a NoSQL data store without any predefined schema. So, the format or data model can be changed any time, without disruption of application. Managing the changes is a difficult problem in SQL.
7. *Support to integrated caching:* NoSQL data store support the caching in system memory. That increases output performance. SQL database needs a separate infrastructure for that.
8. *No inflexibility* unlike the SQL/RDBMS, NoSQL DBs are flexible (not rigid) and have no structured way of storing and manipulating data. SQL stores in the form of tables consisting of rows and columns. NoSQL data stores have flexibility in following ACID rules.

***Types* of*Big Data Problems***

Big Data problems arise due to limitations of NoSQL and other DBs. The following types of problems are faced using Big Data solutions.

1. Big Data need the scalable storage and use of distributed servers together as a cluster. Therefore, the solutions must drop support for the database Joins
2. NoSQL database is open source and that is its greatest strength but at the same time its greatest weakness also because there are not many defined standards for NoSQL data stores. Hence, no two NoSQL data stores are equal. For example:

(i) No stored procedures in MongoDB (NoSQL data store)

(ii) GUI mode tools to access the data store are not available in the market

(iii) Lack of standardization

(iv) NoSQL data stores sacrifice ACID compliancy for flexibility and processingspeed.

## Comparison of NOSQL/RDBMS

| Feature | NOSQL Data Store | SQL/RDBMS |
|---|---|---|
| Model | Schema-less model | Relational |
| Schema | Dynamic schema | Predefined |
| Types of data architecture patterns | Key/value based, column-family based, document based, graph based, object based | Table based |
| Scalable | Horizontally scalable | Vertically scalable |
| Use ofSQL | No | Yes |
| Dataset size preference | Prefers large datasets | Large dataset not preferred |
| Consistency | Variable | Strong |
| Vendor support | Open source | Strong |
| ACID properties | May not support, instead follows Brewer's CAP theorem or BASE properties | Strictly follows |

### 3.5 SHARED-NOTHING ARCHITECTURE FOR BIG DATA TASKS

The columns of two tables relate by a relationship. A relational algebraic equation specifies the relation. Keys share between two or more SQL tables in RDBMS. Shared nothing (SN) is a cluster architecture. A node does not share data with any other node.

Data of different data stores partition among the number of nodes (assigning different computers to deal with different users or queries). Processing may require every node to maintain its own copy of the application's data, using a coordination protocol. Examples are using the partitioning and processing are Hadoop, Flink and Spark.

The features of SN architecture are as follows:

l. *Independence:* Each node with no memory sharing; thus possesses computational self-sufficiency

2. *Self-Healing:* A link failure causes creation of another link

3. *Each node functioning as a shard:* Each node stores a shard (a partition of large DBs)

4. No network contention

## Choosing the Distribution Models

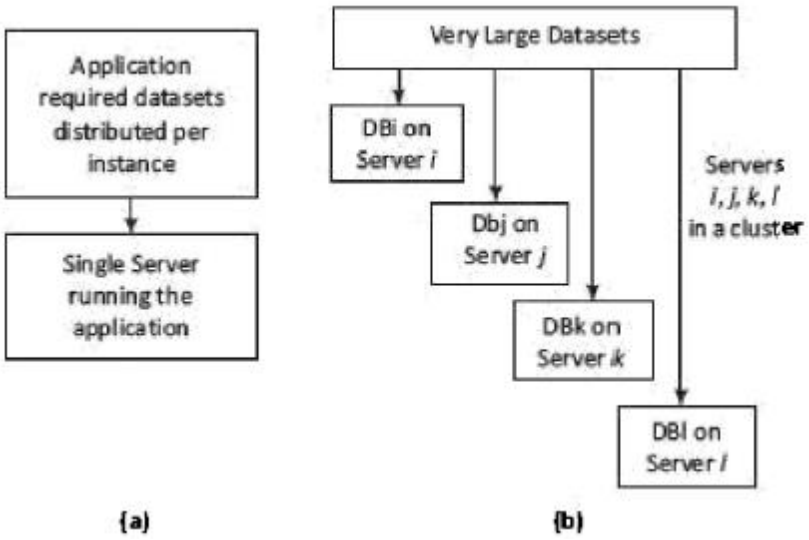
Big Data requires distribution on multiple data nodes at clusters. Distributed software components give advantage of parallel processing; thus providing horizontal scalability. Distribution gives (i) ability to handle large-sized data, and (ii) processing of many read and write operations simultaneously in an application. A resource manager manages, allocates, and schedules the resources of each processor, memory and network connection. Distribution increases the availability when a network slows or link fails. Four models for distribution of the data store are given below:

**Single Server Model**

Simplest distribution option for NoSQL data store and access is Single Server Distribution (SSD) of an application. A graph database processes the relationships between nodes at a server. The SSD model suits well for graph DBs. Aggregates of datasets may be key-value, column-family or BigTable data stores which require sequential processing. These data stores also use the SSD model. An application executes the data sequentially on a single server. Figure 3.9(a) shows the SSD model. Process and datasets distribute to a single server which runs the application.

**Sharding Very Large Databases**

Figure shows sharding of very large datasets into four divisions, each running the application on four i,j, k and l different servers at the cluster. DBi, DBj, DBk and DB1 are four




(a) Single server model (b) Shards distributed on four servers in a cluster

The application programming model in SN architecture is such that an application process runs on

multiple shards in parallel. Sharding provides horizontal scalability. A data store may add an auto-sharding feature. The performance improves in the SN. However, in case of a link failure with the application, the application can migrate the shard DB to another node.

**Master Slave Distribution**

Master directs the slaves. Slave nodes data replicate on multiple slave servers in Master Slave Distribution (MSD) model. When a process updates the master, it updates the slaves also. A process uses the slaves for read operations. Processing performance improves when process runs large datasets distributed onto the slave nodes. Figure 3.10 shows an example of MongoDB. MongoDB database server is mongod and the client is mongo.

**Master-Slave Replication** Processing performance decreases due to replication in MSD distribution model. Resilience for read operations is high, which means if in case data is not available from a slave node, then it becomes available from the replicated nodes. Master uses the distinct write and read paths.

Complexity Cluster-based processing has greater complexity than the other architectures. Consistency can also be affected in case of problem of significant time taken for updating

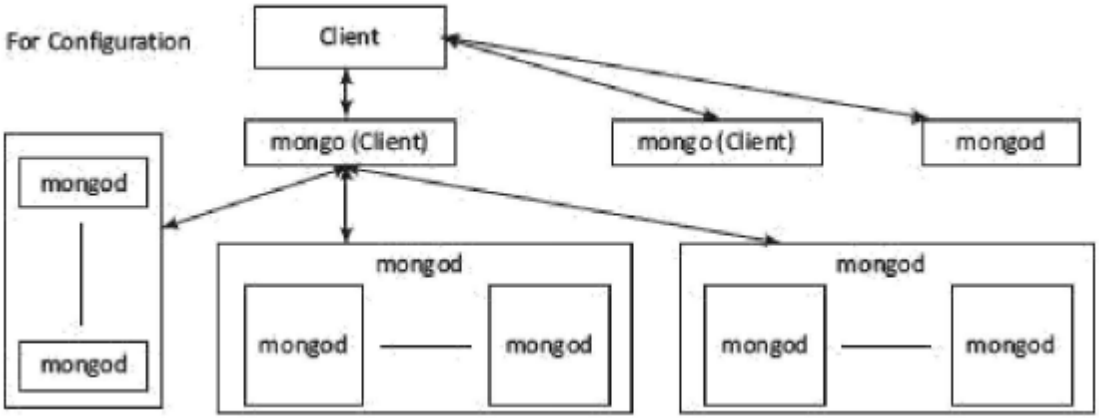



**Figure 3.10 Master-slave distribution model. Mongo is a client and mangod is the server**

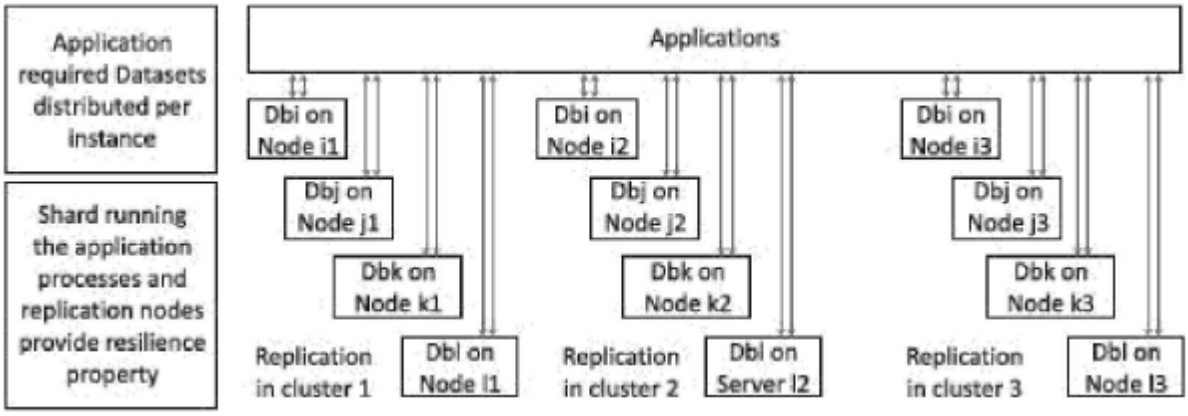
## Peer-to-Peer Distribution Model

Peer-to-Peer distribution (PPD) model and replication show the following characteristics: (1) All replication nodes accept read request and send the responses. (2) All replicas function equally. (3) Node failures do not cause loss of write capability, as other replicated node responds.

Cassandra adopts the PPD model. The data distributes among all the nodes in a cluster.

Performance can further be enhanced by adding the nodes. Since nodes read and write both, a replicated node also has updated data. Therefore, the biggest advantage in the model is

consistency. When a write is on different nodes, then write inconsistency occurs.



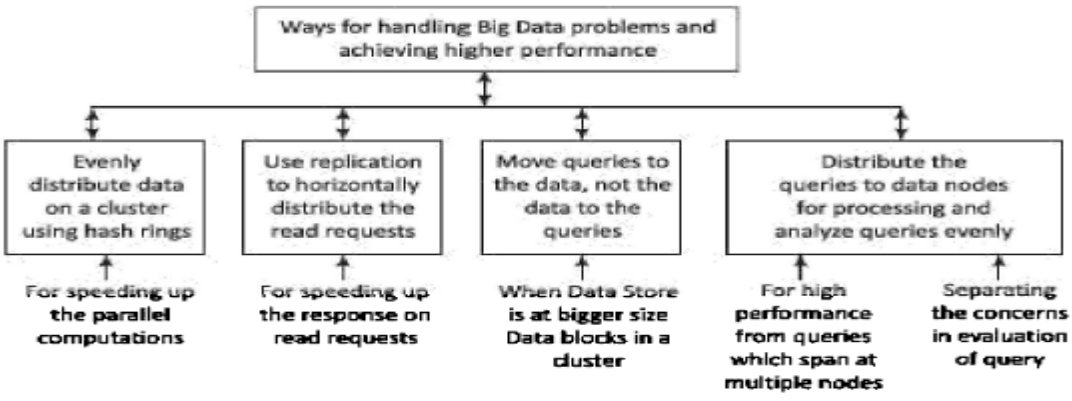

**Shards replicating on the nodes, which does read and writeoperations both**

**Choosing Master-Slave versus Peer-to-Peer**

**Master-slave replication provides** greater scalability for read operations. Replication provides resilience during the read. Master does not provide resilience for writes. Peer-to-peer replication provides resilience for read and writes both.

**Sharing Combining with Replication** Master-slave and sharding creates multiple masters. However, for each data a single master exists. Configuration assigns a master to a group of datasets. Peer-to-peer and sharding use same strategy for the column-family data stores. The shards replicate on the nodes, which does read and write operations both.

**Ways of Handling Big Data Problems**




**Four ways for handling big data problems**

**Following are the ways:**

l. **Evenly distribute the data** on a cluster using the hash rings: Consistent hashing refers to a process where the datasets in a collection distribute using a hashing algorithm which generates the pointer for a collection. Using only the hash of Collection_ID, a Big Data solution client node determines the data location in the cluster. Hash Ring refers to a map of hashes with locations. The client, resource manager or scripts use the hash ring for data searches and Big Data solutions. The ring enables the consistent assignment and usages of the dataset to a specific processor.

**2. Use replication to horizontally** distribute the client read-requests: Replication means creating backup copies of data in real time. Many Big Data clusters use replication to make the failure-proof retrieval of data in a distributed environment. Using replication enables horizontal scaling out of the client requests.

**3. Moving queries to the data**, not the data to the queries: Most NoSQL data stores use cloud utility services (Large graph databases may use enterprise servers). Moving client node queries to the data is efficient as well as a requirement in Big Data solutions.

**4. Queries distribution to multiple nodes:** Client queries for the DBs analyze at the analyzers, which evenly distribute the queries to data nodes/ replica nodes. High performance query processing requires usages of multiple nodes. The query execution takes place separately from the query evaluation (The evaluation means interpreting the query and generating a plan for its execution sequence).

## 3.6 MONGODB DATABASE

MongoDB is an open source DBMS. MongoDB programs create and manage databases. MongoDB manages the collection and document data store. MongoDB functions do querying and accessing the required information. The functions include viewing, querying, changing, visualizing and running the transactions. Changing includes updating, inserting, appending or deleting.

MongoDB is (i) non-relational, (ii) NoSQL, (iii) distributed, (iv) open source, (v) document based (vi) cross-platform, (vii) Scalable, (viii) flexible data model, (ix) Indexed, (x) multi-master and (xi) fault tolerant. Document data store in SON-like documents. The data store uses the dynamic schemas.

The typical MongoDB applications are content management and delivery systems, mobile

applications, user data management, gaming, e-commerce, analytics, archiving and logging.

Features of Mango D B

*MongoDB data store* is a physical container for collections. Each DB gets its own set of files on the file system. A number of DBs can run on a single MongoDB server. DB is default DB in MongoDB that stores within a data folder. The database server ofMongoDB is *mongod* and the client is *mongo.*

2. *Collection* stores a number of MongoDB documents. It is analogous to a table of RDBMS. A collection exists within a single DB to achieve a singlepurpose. Collections may store documents that do not have the same fields. Thus, documents of the collection are schema-less. Thus, it is possible to store documents of varying structures in a collection. Practically, in an RDBMS, it is required to define a column and its data type, but does not need them while working with the MongoDB.
3. *Document model* is well defined. Structure of document is clear, Document is the unit of storing data in a MongoDB database. Documents are analogousto the records of RDBMS table. Insert, update and delete operations can beperformed on a collection. Document use *JSON* OavaScript Object Notation) approach for storing data. *JSON* is a lightweight, self-describing format used to interchange data between various applications. JSON data basically has key-value pairs. Documents have dynamic schema.
4. MongoDB is a document data store in which one collection holds differentdocuments. Data store in the form of ]SON-style documents. Number of fields, content and size of the document can differ from one document to another.
5. *Storing of data* is flexible, and data store consists of JSON-like documents. This implies that the fields can vary from document to document and data structure can be changed over time; *JSON* has a standard structure, and scalable way of describing hierarchical data (Example 3.3(ii)).
6. *Storing of documents* on disk is in BSON serialization format. BSON is a binary representation of JSON documents. The mongo JavaScript shell and MongoDB language drivers perform translation between BSON and language-specific document representation.
7. *Querying, indexing, and real time aggregation* allows accessing and analyzing the data efficiently.

8. *Deep query-ability-Supports* dynamic queries on documents using adocument-based query language that's nearly as powerful as SQL.
9. No complexJoins.
10. *Distributed DB* makes availability high, and provides horizontal scalability.
11. *Indexes on any field* in a collection of documents: Users can create indexes on any field in a document. Indices support queries and operations. By default, MongoDB creates an index on the _id field of every collection.
12. *Atomic operations on a single document* can be performed even though support of multi-document transactions is not present. The operations are alternate to ACID transaction requirement of a relational DB.
13. *Fast-in-place updates:* The DB does not have to allocate new memory location and write a full new copy of the object in case of data updates. This results into high performance for frequent update use cases. For example, incrementing a counter operation does not fetch the document from the server. Here, the increment operation can simply be set.
14. *No configurable cache:* MongoDB uses all free memory on the system automatically by way of memory-mapped files (The operating systems use the similar approach with their file system caches). The most recently used data is kept in RAM. If indexes are created for queries and the working dataset fits in RAM, MongoDB serves all queries from memory.
15. *Conversion/mapping* of application objects to data store objects not needed

**Dynamic Schema** Dynamic schema implies that documents in the same collection do not need to have the same set of fields or structure. Also, the similar fields in a document may contain different types of data. Table 3.8 givesthe comparison with RDBMS

| RDBMS | MongoDB |
|---|---|
| Database | Data store |
| Table | Collection |
| Column | Key |
| Value | Value |
| Records / Rows / Tuple | Document/ Object |
| Joins | Embedded Documents |
| Index | Index |
| Primary key | Primary key (_id) is default key provided by MongoDB itself |

**Comparison of Mango DB and RDBMS**

**Replication:** Replication ensures high availability in Big Data. Presence of multiple copies increases on different database servers. This makes DBs fault- tolerant against any database server failure. Multiple copies of data certainly help in localizing the data and ensure availability of data in a distributed system environment.

**MongoDB replicates** with the help of a replica set. A replica set in MongoDB is a group of mongod (MongoDb server) processes that store the same dataset. Replica sets provide redundancy but high availability. A replica set usually has minimum three nodes. Any one out of them is called primary. The primary node receives all the write operations. All the other nodes are termed as secondary. The data replicates from primary to secondary nodes. A new primary node can be chosen among the secondary nodes at the time of automatic failover or maintenance. The failed node when recovered can join the replica set as secondary node again.

| Commands | Description |
|---|---|
| rs.initiate() | To initiate a new replica set |
| rs.conf () | To check the replica set configuration |
| rs.status () | To check the status of a replica set |
| rs.add () | To add members to a replica set |

S

Figure shows a replicated dataset after creating three secondary members from a primary member.

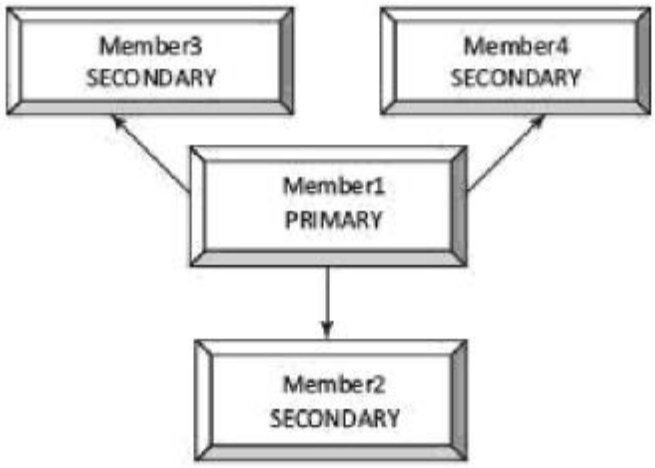



**Figure 3.13** Replicated set on creating secondary members

**Auto-sharding** :Sharding is a method for distributing data across multiple machines in a distributed application environment. MongoDB uses sharding to provide services to Big Data

applications.

A single machine may not be adequate to store the data. When the data size increases, do not provide data retrieval operation. Vertical scaling by increasing the resources of a single machine is quite expensive. Thus, horizontal scaling of the data can be achieved using sharding mechanism where more database servers can be added to support data growth and the demands of more read and write operations.

Sharding automatically balances the data and load across various servers. Sharding provides additional write capability by distributing the write load over a number of mongod (MongoDB Server) instances.

| Type | Description |
|---|---|
| Double | Represents a float value. |
| String | UTF-8 format string. |
| Object | Represents an embedded document. |
| Array | Sets or lists of values. |
| Binary data | String of arbitrary bytes to store images, binaries. |
| Object id | Objectlds (MongoDB document identifier, equivalent to a primary key) are: small, likely unique, fast to generate, and ordered. The value consists of 12-bytes, where the first four bytes are for timestamp that reflects the instance when Objectld creates. |
| Boolean | Represents logical true or false value. |
| Date | BSON Date is a 64-bit integer that represents the number of milliseconds since the Unix epoch Oan 1, 1970). |
| Null | Represents a null value. A value which is missing or unknown is Null. |
| Regular Expression | RegExp maps directly to aJavaScript RegExp |
| 32-bit integer | Numbers without decimal points save and return as 32-bit integers. |
| Timestamp | A special timestamp type for internal MongoDB use and is not associated with the regular date type. Timestamp values are a 64-bit value, where first 32 bits are time, t (seconds since the Unix epoch), and next 32 bits are an incrementing ordinal for operations within a given second. |
| 64-bit integer | Number without a decimal point save and return as 64-bit integer . |

| | |
|---|---|
| Min key | MinKey compare less than all other possible BSON element values, respectively, and exist primarily for internal use. |
| Max key | MaxKey compares greater than all other possible BSON element values, respectively, and exist primarily for internal use. |

**Data Types which Mango DB document Supports**

**Rich Queries and Other DB Functionalities** MongoDB offers a rich set of features and functionality compared to those offered in simple key-value stores. They can be comparable to those offered by any RDBMS. MongoDB has a complete query language, highly-functional secondary indexes (including text search and geospatial), and a powerful aggregation framework for data analysis. MongoDB provides functionalities comparison of features.

| Features | RDBMS | MongoDB |
|---|---|---|
| Rich Data Model | No | Yes |
| Dynamic Schema | No | Yes |
| Typed Data | Yes | Yes |
| Data Locality | No | Yes |
| Field Updates | Yes | Yes |
| Complex Transactions | Yes | No |
| Auditing | Yes | Yes |
| Horizontal Scaling | No | Yes |

**Comparison of features MongoDB with respect to RDBMS**

| Command | Functionality |
|---|---|
| Mongo | Starts MongoDB; (*mongo is MongoDB client). The defaultdatabase **in** MongoDB is test. |
| db.help() | Runs help. This displays the list of all the commands. |
| db.stats() | Gets statistics about MongoDB server. |

| | |
|---|---|
| Use <database name) | Creates database |
| Db | Outputs the names of existing database, if created earlier |
| Dbs | Gets list of all the databases |
| db.dropDatabase () | Drops a database |
| db.database name.insert () | Creates a collection using insert () |
| db.<database name>. find() | Views all documents in a collection |
| db.<database name>.update () | Updates a document |
| db.<database name>.remove () | Deletes a document |

**MongoDB querying commands**

Following explains the sample usages of the commands:

**To Create database** Command use - use command creates a database; For example, Command use lego creates a database named lego. (A sample database is created to demonstrate subsequent queries. The Lego is an international toy brand). Default database in MongoDB is test.

**To see the existence of database Command db** - db command shows that lego database is created.

**To get list of all the databases Command show dbs** - This command shows the names of all the databases.

**To drop database Command db. dropDatabase () -** This command drops a database. Run use lego command before the db. dropDatabase () command to drop lego Database. If no database is selected, the default database test will be dropped.

**To create a collection Command insert ()** -Tocreate a collection, the easiest way is to insert a record (a document consisting of keys (Field names) and Values) into a collection. A new collection will be created, if the collection does not exist. The following statements demonstrate the creation of a collection with three fields (ProductCategory, Productld and ProductName) in the lego:

```
db.lego.insert
(
        {
                "ProductCategory": "Airplane",
                "ProductId": 10725,
                "ProductName": "Lost Temple"
        }
)
```

**To view all documents in a collection Command db. <database name>.** find ()-Find command is equivalent to select query of RDBMS. Thus, "Select * from lego" can be written as db. lego. find () in MongoDB. MongoDB created unique objecteld ("_id") on its own. This is the primary key of the collection. Command db. <database name>. find() .pretty() gives a prettier look.

**To update a document Command db. <database name>.** update ()-Update command is used to change the field value. By default, multi attribute is false. If

{multi: true} is not written then it will update only the first document.

**To delete a document Command db. <database name>.** remove () - Remove command is used to delete the document. The query db. <database name>. remove ( ("ProdctID": 10725)) removes the document whose productld is 10725.

**To add array in a collection Command insert ()** - Insert command can also be used to insert multiple documents into a collection at one time.

```
db.lego.insert
(
        [
                {
                        "ProductCategory": "Airplane",
                        "ProductId": 10725,
                        "ProductName": "Lost Temple"
                },
                {
                        "ProductCategory": "Airplane",
                        "ProductId": 31047,
                        "ProductName": "Propeller Plane"
                },
                {
                        "ProductCategory": "Airplane",
                        "ProductId": 31049,
                        "ProductName": "Twin Spin Helicopter"
                }
        ]
)
```

## CASSANDRA DATA BASE

Cassandra was developed by Facebook and released by Apache. Cassandra was named after

Trojan mythological prophet Cassandra, who had classical allusions to a curse on oracle. Later on, IBM also released the enhancement of Cassandra, as open

source version. The open source version includes an IBM Data Engine which processes No SQL data store. The engine has improved throughput when workload of read-operations is intensive.

Cassandra is basically a column family database that stores and handles massive data of any format including structured, semi-structured and unstructured data.

Apache Cassandra DBMS contains a set of programs. They create and manage databases. Cassandra provides functions (commands) for querying the data and accessing the required information. Functions do the viewing, querying and changing (update, insert or append or delete), visualizing and perform transactions on the DB.

Apache Cassandra has the distributed design of Dynamo. Cassandra is written in Java. Big organizations, such as Facebook, IBM, Twitter, Cisco, Rackspace, eBay, Twitter and Netflix have adopted Cassandra.

Characteristics of Cassandra are (i) open source, (ii) scalable (iii) non- relational (v) NoSQL (iv) Distributed (vi) column based, (vii) decentralized, (viii) fault tolerant and (ix) tuneable consistency.

Features of Cassandra are as follows:

1. Maximizes the number of writes - writes are not very costly (time consuming)
2. Maximizes data duplication
3. Does not support Joins, group by, OR clause and aggregations
4. Uses Classes consisting of ordered keys and semi-structured data storage systems
5. Is fast and easily scalable with write operations spread across the cluster. The cluster does not have a master-node, so any read and write can be handled by any node in the cluster.
6. Is a distributed DBMS designed for handling a high volume of structured data across multiple cloud servers

Has peer-to-peer distribution in the system across its nodes, and the data is distributed among all the nodes in a cluster.

**Data Replication** Cassandra stores data on multiple nodes (data replication) and thus has no single point of failure, and ensures availability, a requirement in CAP theorem. Data replication uses a replication strategy. Replication factor determines the total number of

replicas placed on different nodes. Cassandra returns the most recent value of the data to the client. If it has detected that some of the nodes responded with a stale value, Cassandra performs a read repair in the background to update the stale values.

**Components at Cassandra** Table 3.13 gives the components at Cassandra and their description

| Component | Description |
|---|---|
| Node | Place where data stores for processing |
| Data Center | Collection of many related nodes |
| Cluster | Collection of many data centers |
| Commit log | Used for crash recovery; each write operation written to commit log |
| Mem-table | Memory resident data structure, after data written in commit log, data write in mem-table temporarily |
| SSTable | When mem-table reaches a certain threshold, data flush into an SSTable disk file |
| Bloom filter | Fast and memory-efficient, probabilistic-data structure to find whether an element is present in a set, Bloom filters are accessed after every query. |

**Scalability** Cassandra provides linear scalability which increases the throughput and decreases the response time on increase in the number of nodes at cluster.

**Transaction Support** Supports ACID properties (Atomicity, Consistency, Isolation, and Durability).

**Replication Option** Specifies any of the two replica placement strategy names. The strategy names are Simple Strategy or Network Topology Strategy. The replica placement strategies are:

**Simple Strategy:** Specifies simply a replication factor for the cluster.

**Network Topology Strategy:** Allows setting the replication factor for each data center independently.

Table 3.14 Data types built into Cassandra, their usage and description

| CQL Type | Description |
|---|---|
| ascii | US-ASCII character string |
| bigint | 64-bit signed long integer |
| blob | Arbitrary bytes (no validation), BLOB expressed in hexadecimal |
| boolean | True or false |
| counter | Distributed counter value (64-bit long) |

| | |
|---|---|
| decimal | Variable-precision decimal integer, float |
| double | 64-bit IEEE-754 *double precession* floating point integer, float |
| float | 32-bit IEEE-754 single precession floating point integer, float |
| inet | IP address string in 1Pv4 or 1Pv6 format, used by the python-cql driver and CQL native protocols |
| int | 32-bit signed integer |
| list | A collection of one or more ordered elements |
| map | AJSON-style array of literals: {literal: literal, literal: literal ...} |
| set | A collection of one or more elements |
| text | UTF-8 encoded string |
| timestamp | Date plus time, encoded as 8 bytes since epoch integers, strings |
| varchar | UTF-8 encoded string |
| varint | Arbitrary-precision integer |

Cassadra Data Model Cassandra Data model is based on Google's BigTable Each value maps with two strings (row key, column key) and timestamp, similar to HBase.The database can be considered as a sparse distributed multi-dimensional sorted map. Google file system splits the table into multiple tablets (segments of the table) along a row. Each tablet, called METAl tablet, maximum size is 200 MB, above which a compression algorithm used. META0 is the master-server. Querying by META0 server retrieves a METAl tablet. During execution of the application, caching of locations of tablets reduces the number of queries.

**Cassandra Data Model consists of four main components**: (i) Cluster: Made up of multiple nodes and keyspaces, (ii} Keyspace: a namespace to group multiple column families, especially one per partition,

**Column:** consists of a column name, value and timestamp and (iv) Column- family: multiple columns with row key reference. Cassandra does keyspace management using partitioning of keys into ranges and assigning different key- ranges to specific nodes.

Following Commands prints a description (typically a series of DDL statements) of a schema element or the cluster:

```
DESCRIBE CLUSTER
DESCRIBE SCHEMA
```

DESCRIBE KEYSPACES

DESCRIBE KEYSPACE <keyspace name>

DESCRIBE TABLES

DESCRIBE TABLE <table name>

DESCRIBE INDEX <index name>

DESCRIBE MATERIALIZED VIEW <view name>DESCRIBE

TYPES

DESCRIBE TYPE <type name>

DESCRIBE FUNCTIONS

DESCRIBE FUNCTION <function name>DESCRIBE

AGGREGATES

DESCRIBE AGGREGATE <aggregate function name>

Consistency Command CONSISTENCY shows the current consistency level. CONSISTENCY <LEVEL> sets a new consistency level. Valid consistency levels are ANY, ONE, TWO, THREE,QUORUM, LOCAL_ONE, LOCAL_QUORUM, EACH_QUORUM, SERIAL AND LOCAL_SERIAL. Following are their meanings:

1. ALL: Highly consistent. A write must be written to commitlog andmemtable on all replica nodes in the cluster.
2. EACH_QUORUM: A write must be written to commitlog and memtable onquorum of replica nodes in all data centers.
3. LOCAL_QUORUM: A write must be written to commitlog and memtable onquorum of replica nodes in the same center.
4. ONE: A write must be written to commitlog and memtable of at least onereplica node.
5. TWO, THREE: Same as One but at least two and three replica nodes, respectively.
6. LOCAL_ONE: A write must be written for at least one replica node in the local data center.
7. ANY: A write must be written to at least one node.
8. SERIAL: Linearizable consistency to prevent unconditional update.
9. LOCAL_SERIAL: Same as Serial but restricted to the local data center.

**Keyspaces** A keyspace (or key space) in a NoSQL data store is an object that contains all column families of a design as a bundle. Keyspace is the outermostgrouping of the data in the data store. It is similar to relational database. Generally, there is one keyspace per application. Keyspace in Cassandra is a namespace that defines data replication on nodes. A cluster contains one keyspace per node.

Create Keyspace Command CREATE KEYSPACE <Keyspace Name> WITH replication = {'class': '<Strategy name>', 'replication_factor': '<No. of replicas>'}AND durable_writes= '<TRUE/FALSE>';

CREATE KEYSPACE statement has attributes replication with option class and replication factor, and durable_write.

Default value of *durable_ writes* properties of a table is set to true. That commands the Cassandra to use Commit Log for updates on the current Keyspace true or false. The option is not compulsory.

1. ALTER KEYSPACE command changes (alter) properties, such as the numberof replicas and the durable_writes of a keyspace: ALTER KEYSPACE <Keyspace Name> WITH replication = {'class': '<Strategy name>', 'replication_factor':'<No. of replicas>'};
2. DESCRIBE KEYSPACE command displays the existing keyspaces.
3. DROP KEYSPACE command drops a keyspace:
4. Re-executing the drop command to drop the same keyspace will result in configuration exception.
5. Use KEYSPACE command connects the client session with a keyspace.

| Command | Functionality |
|---|---|
| CQLSH | A command line shell for interacting with Cassandra throughCQL |
| HELP | Runs help. This displays the list of all the commands |
| CONSISTENCY | Shows the current consistency level |
| EXIT | Terminate the CQL shell |
| SHOW HOST | Displays the host |

| | |
|---|---|
| SHOW VERSION | Displays the details of current cqlsh session such as host,Cassandra version, or data type assumptions |
| CREATE KEYSPACE <Keyspace Name> | Creates keyspace with a name |
| DESCRIBE KEYSPACE <Keyspace Name> | Displays the keyspace with a name |
| ALTER KEYSPACE <Keyspace Name> | Modifies keyspace with a name |
| DROP KEYSPACE <Keyspace Name> | Deletes keyspace with a name |
| CREATE (TABLE \| COLUMNFAMILY) | Creates a table or column family |
| COLLECTIONS | Lists the Collections |

**CQL commands and their functionalities**

Give the examples of usages of various CQL commands.

**SOLUTION**

(1) Create Table Command: CREATE TABLE command creates a table in the current keyspace:

CREATE (TABLE COLUMNFAMILY) <tablename> ('<column-definition>', '<column-definition>')(WITH <option> AND <option>);

***Primary key* is a column used to uniquely identify a row. Therefore, defining a primary key is compulsory while creating a table. A primary key is made of one or more columns of a table.**

***Example:* Create a table *Productinfo* in the keyspace *lego*, with primary key field *Productid*.**

Use lego;

Create table Productinfo(Productid int primarykey, ProductType text);

**(2) Describe Tables Command: DESCRIBE TABLE Command displays all the tables in the current keyspace:**

DESCRIBE TABLE <TABLE NAME>;

***Example:* Display the details of a table Productinfo:**

DESCRIBE TABLE Productinfo;

**(3) Alter Tables Command:**

ALTER TABLE Command ALTER (TABLE COLUMNFAMILY) <tablename> (ADD I DROP) <column name>

**(4) Cassandra CURD Operations: (CURD-Create, Update, Read and Delete data into tables)** :

**(a) Insert Command:**

INSERT INTO <tablename> (<columnl name>, <column2name>....) VALUES (<valuel>, <value2>....) USING

<option>

**(a) Update Command:**

**UPDATE command updates data in a table. The following keywords are used while updating data in a table:**

**Where - This clause is used to select the row to be updated.**

**Set - Set the value using this keyword.**

**Must- Includes all the columns composing the primary key.**

**If a given row is unavailable, then UPDATE creates a new row.**

UPDATE <tablename> SET <column name>= <new value> <column name>= <value>.... WHERE <condition>

**(a) Select Command**

**SELECT command reads the data from a table. The command can read a whole table, a single column, or a particular cell:**

SELECT <column name(s)> FROM <Table Name>

**To select all records:**

SELECT* FROM <Table Name>

**To select records that fulfils required condition:**

SELECT <columnl, column2,..> FROM <Table Name>where <Condition>

(b) **Delete Command**

**DELETE command deletes data from a table:**

DELETE FROM <identifier> WHERE <condition>; ***Example:*** **Delete row from a table where Product id is 31047:** DELETE FROM Productinfo WHERE Productid = 31047;

**(5) Creating a Table with List**

**CREATE Table command is used for creating a table with a list.**

**The following query creates a table with two columns, one is the primary key and the other has multiple items (List):**

CREATE TABLE data (<column name>, <data type>PRIMARY KEY, <column name list<data type>);

***Example*** : **Create a sample table *Contactlnfo* with three columns: *Sno, name* and *Emailld.* To store multiple Email Ids, use a list:**

create table Contactinfo (Sno int Primary key,Name text, emailid list <text>);

(6) Update Command for updating Data into a List

UPDATE command also updates data into a list:

UPDATE <table Name> SET <New data> where <condition>.

*Example* : Add one more email Id to the *emailld* list in *Contactlnfo* table :

UPDATE Contactinfo SET emailid = emailid + ['preeti@ymail.com'] where SNo=l.

# MODULE 4

# MapReduce, Hive and Pig

*Syllabus to Discuss*

MapReduce, Hive and Pig: Introduction, MapReduce Map Tasks, Reduce Tasks and MapReduce Execution, Composing MapReduce for Calculations and Algorithms, Hive, HiveQL, Pig.

## INTRODUCTION

The data processing layer is the application support layer, while the application layer is the data consumption layer in Big-Data architecture design, when using HDFS, the Big Data processing layer includes the API's of Programs such as **MapReduce** and **Spark.**

- ✓ The application support layer includes HBase which creates column-family data store using other formats such as key-value pairs or JSON file.
- ✓ HBase stores and processes the columnar data after translating into MapReduce tasks to run in HDFS.
- ✓ The support layer also includes Hive which creates SQL-like tables. Hive stores and processes table data after translating it into MapReduce tasks to run in HDFS.
- ✓ Hive creates SQL-like tables in Hive shell. Hive uses HiveQL processes queries, ad hoc (unstructured) queries, aggregation functions and summarizing functions, such as functions to compute maximum, minimum, average of selected or grouped datasets. HiveQL is a restricted form of SQL.
- ✓ The support layer also includes Pig. Pig is a data-flow language and an execution framework.
- ✓ Pig enables the usage of relational algebra in HDFS. MapReduce is the processing framework and YARN is the resource managing framework.

Figure 4.1 shows Big Data architecture design layers: (i) data storage, (ii) data processing and data consumption, (iii) support layer APis for MapReduce, Hive and Pig running on top of the HDFS Data Store, and (v) application tasks. Pig is a dataflow language, which means that it defines a data stream and a series of transformations.

The smallest unit of data that can be stored or retrieved from the disk is a block. HDFS deals with the data stored in blocks.

The Hadoop application is responsible for distributing the data blocks across multiple nodes. The tasks, therefore, first convert into map and reduce tasks.

This requirement arises because the mapping of stored values is very important. The number of map tasks in an application is handled by the number of blocks of input files.

Reduce task uses those values for further processing such as counting, sorting or aggregating.

Application sub-task assigned for processing needs only the outputs of reduce tasks. For example, a query needs the required response for a data store.

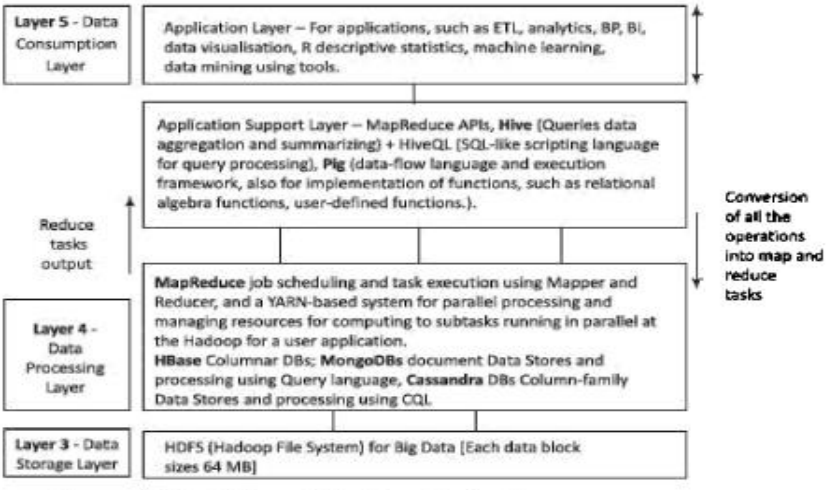



**Figure 4.1** Big Data architecture design layers

***MapReduce programming model*** refers to a programming paradigm for processing Big Data sets with a parallel and distributed environment using map and reduce tasks.

***YARN*** refers to provisioning of running and scheduling parallel programs for map and reduce tasks and allocating parallel processing resources for computing sub-tasks running in parallel at the Hadoop for a user application.

The YARN resources management enables large-scale data analytics using multiple machines (data nodes) in the HDFS cluster.

***Script*** refers to a small program (codes up to few thousand lines of code) in a language used for purposes such as query processing, text processing, or refers to a small code written in a dynamic high-level general-purpose language, such as Python or PERL.

***SQL-like scripting language*** means a language for writing script that processes queries similar to SQL. SQL lets us: (i) write structured queries for processing in DBMS, (ii) create and modify schema, and control the data access, (iii) Create client for sending query scripts, and create and manage server databases, and (iv) view, query and change (update, insert or append or delete)databases.

A theorem known as CAP (Consistency, Availability and £,artitions) states that out of three properties, at least two must be present for the application/service/process.

NoSQL relies upon another model known as the BASE model. This model has three principles: Basic availability (the availability of data even in the presence of multiple failures), Soft state (data consistencyis the developer's problem and should not be handled by the database).

Eventual consistency (when no new changes occur on existing data, eventually all accesses to that data will return the last updated value).

***Data-architecture patterns*** refer to formats used in NoSQL DBs. The examples are Key-Value Data Stores, Object Data Stores, Column family Big Data Stores, Tabular Data Stores and Document Stores.

***Key-Value Data Store*** refers to a simplest way to implement a schema-less database. A string called key maps to values in a large data string or BLOB (basic large object)..

***Object Data Store*** refers to a repository which stores the (i) objects (such as files, images, documents, folders and business reports), (ii) system metadata which provides information such as filename, creation_date, last_modified, language_used, access_permissions, supported Query languages, and (iii) Custom metadata which provides information such as subject, category and sharing permission.

**Tabular Data Store refers to table, column-family or BigTable like Data Store**.

***Column family Big Data*** store refers to storage in logical groups of column families. The storage may be similar to columns of sparse matrix. They use a pair of row and column keys to access the column fields.

***BigTable Data Store*** **is a popular column-family based Data Store.**

Row key,column key and timestamp uniquely identify a value. Google BigTable, HBaseand Cassandra DBs use the BigTable Data Store model.

***Document Store*** means a NoSQL DB which stores hierarchical information in a single unit called document. Document stores data in nested hierarchies, for example in XML document object model, JSON formats data model or machine-readable data as one BLOB.

***Tuple*** means an ordered list of elements. An n-tuple relates to set theory, a collection (sequence) of "n" elements. Tuples implement the records.

***Collection*** means a well-defined collection of distinct objects in a set, the objects of a set are the elements. A collection may be analogous to a table of RDBMS. A collection in a database also refers to storage of a number of documents.

***Aggregate*** refers to collection of data sets in the key value, column family or BigTable data stores which usually require sequential processing.

***Aggregation function*** refers to a function to find counts, sum, maximum, minimum, other statistical or mathematical function using a collection of datasets, such as column or column-family.

**Sequence** refers to an enumerated collection of objects, (the repetitions can be there) which contain members similar to a set. Sequence length equals the number of elements (can also be infinite). Sequence should reflect an order which matters, unlike a set.

***Document*** refers to a container for the number of collections. The container can be a unit of storing data in a database, such as MongoDB.

***Natural join*** is where two tables join based on all common columns. Both thetables must have the same column name and the data type.

## MAPREDUCE MAP TASKS, REDUCE TASKS ANDMAPREDUCE EXECUTION

Big Data Processing employs the Map Reduce Programming Model. A job means a Map Reduce Program. Each job consists of several smaller unit, called MapReduce Tasks.

A software execution framework in MapReduce programming defines the parallel tasks.

The Hadoop MapReduceimplementation usesJava framework.

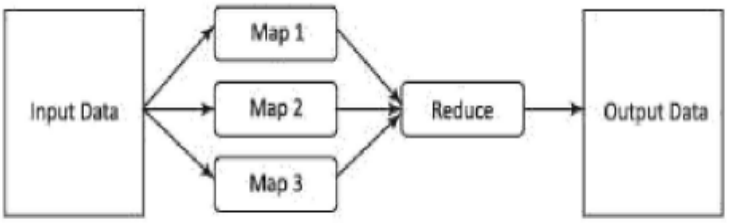



**Figure 4.2** MapReduce Programming Model

The model defines two important tasks, namely Map and Reduce.

Map takes input data set as pieces of data and maps them on various nodes for parallel processing.

The reduce task, which takes the output from the maps as an input and combines those data pieces into a smaller set of data. A reduce task always run after the map task (s).

Many real-world situations are expressible using this model.

**I*nner join*** is the default natural join. It refers to two tables that join based on common columns mentioned using the ON clause. Inner Join returns all rows from both tables if the columns match.

*Node* refers to a place for storing data, data block or read or write computations.

***Data center*** in a DB refers to a collection of related nodes. Many nodes form a data center or rack.

***Cluster*** refers to a collection of many nodes.

***Keyspace*** means a namespace to group multiple column families, especially one per partition.

***Indexing to a field*** means providing reference to a field in a document of collections that support the queries and operations using that index. A DB creates an index on the _id field of every collection.

The input data is in the form of an HDFS file. The output of the task also gets stored in the HDFS.

The compute nodes and the storage nodes are the same at a cluster, that is, the MapReduce program and the HDFS are running on the same set of nodes.

.

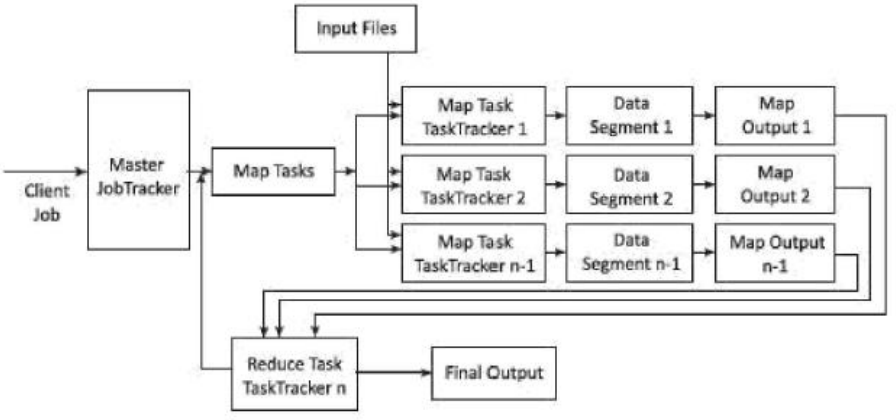



**Figure 4.3** MapReduce process on client submitting a job

Figure 4.3 shows MapReduce process when a client submits a job, and the succeeding actions by the JobTracker andTaskTracker.

**JobTracker and Task Tracker** MapReduce consists of a single master JobTracker and one slave TaskTracker per cluster node.

The **master** is responsible for scheduling the component tasks in a job onto the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master.

The data for a MapReduce task is initially at input files. The input files typically reside in the HDFS. The files may be line-based log files, binary format file, multi-line input records, or something else entirely different.

The MapReduce framework operates entirely on key, value-pairs. The framework views the input to the task as a set of (key, value)pairs and produces a set of (key, value) pairs as the output of the task, possiblyof different types.

## Map-Tasks

***Map task*** means a task that implements a map(), which runs user application codes for each key-value pair **(kl, vl).** Key **kl** is a set of keys. Key **kl** maps to group of data values (Section 3.3.1). Values **vl** are a large string which is read from the input file(s).

The **output** of map() would be zero (when no values are found) or intermediate key-value pairs **(k2, v2).** The value v2 is the information for the transformation operation at the reduce task using aggregation or other reducing functions.

***Reduce task*** refers to a task which takes the output v2 from the map as an input and combines those data pieces into a smaller set of data using a *combiner*. The reduce task is always performed after the map task.

The ***Mapper*** performs a function on individual values in a dataset irrespective of the data size of the input. That means that the Mapper works on a single data set. Figure 4.4 shows logical view of functioning of map().

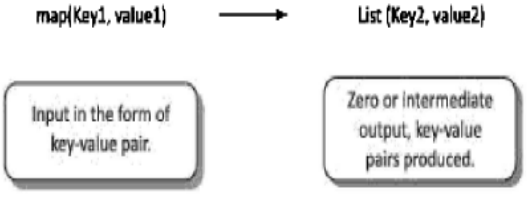



**Figure 4.4** Logical view of functioning of map()

Hadoop Java API includes Mapper class. An abstract function map() is present in the Mapper class. Any specific Mapper implementation should be a subclass of this class and overrides the abstract function, map ().

## The Sample Code for Mapper Class

**public clase SampleMapper extends Mapper<kl, Vl, k2, *v2*>**

{

**void map (kl key, Vl value, Context context) throwe IOException, InterruptedException**

{..}

Individual Mappers do not communicate with each other.

**Number of Maps** The number of maps depends on the size of the input files, i.e., the total number of blocks of the input files.

If the input files are of 1TB in size and the block size is 128 MB, there will be 8192 maps. The number of map task Nmap can be explicitly set by using *setNumMapTasks(int).* Suggested numberis nearly 10-100 maps per node. Nmap can be set even higher.

## Key-Value Pair

Each phase (Map phase and Reduce phase) of MapReduce has key-value pairs as input and output. Data should be first converted into key-value pairs before it ispassed to the Mapper, as the Mapper only understands key-value pairs of data.

*Key-value pairs in Hadoop MapReduce are generated as follows:*

***InputSplit*** - Defines a logical representation of data and presents a Split data for processing at individual map().

***RecordReader*** - Communicates with the InputSplit and converts the Split into

records which are in the form of key-value pairs in a format suitable for readingby the Mapper.

***RecordReader uses TextInputFormat*** by default for converting data into key-value pairs.

***RecordReader*** communicates with the InputSplit untilthe file is read.

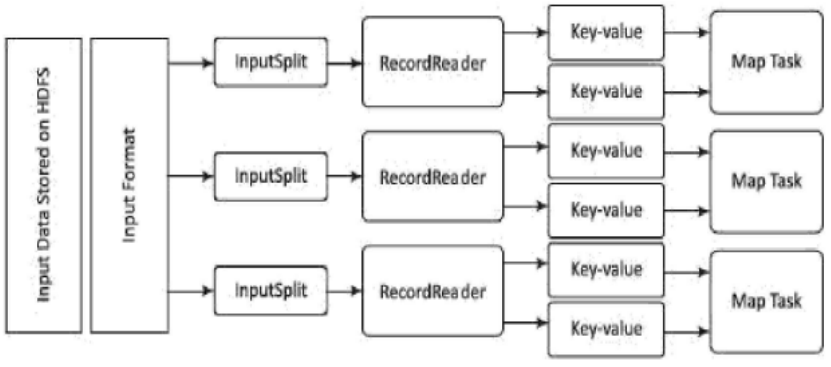



**Figure 4.5** Key-value pairing in MapReduce

Figure 4.5 shows the steps in MapReduce key-value pairing.

Generation of a key-value pair in MapReduce depends on the dataset and the required output. Also, the functions use the key-value pairs at four places: map() input, map() output, reduce() input and reduce() output.

## Grouping by Key

When a map task completes, Shuffle process aggregates (combines) all the Mapper outputs by grouping the key-values of the Mapper output, and the value v2 append in a list of values. A "Group By" operation on intermediate keys creates **v2.**

### *Shuffle and Sorting Phase*

All pairs with the same group key **(k2)** collect and group together, creatingone group for each key.

Shuffle output format will be a List of **<k2,** List **(v2)>.** Thus, a different subset of the intermediate key space assigns to each reduce node.

These subsets of the intermediate keys (known as "partitions") are inputs to the reduce tasks.

Each reduce task is responsible for reducing the values associated with partitions. HDFS sorts the partitions on a single node automatically before they input to the Reducer.

## Partitioning

- ✓ The Partitioner does the partitioning. The partitions are the semi-mappers in MapReduce.
- ✓ Partitioner is an optional class. MapReduce driver class can specify the Partitioner.
- ✓ A partition processes the output of map tasks before submitting it to Reducer tasks.
- ✓ Partitioner function executes on each machine that performs a map task.
- ✓ Partitioner is an optimization in MapReduce that allows **local partitioning** before reduce-task phase.
- ✓ The same codes implement the Partitioner, Combiner as well as reduce() functions.
- ✓ Functions forPartitioner and sorting functions are at the mapping node.
- ✓ The main function of a Partitioner is to split the map output records with the same key.

## Combiners

Combiners are semi-reducers in MapReduce. Combiner is an optional class. MapReduce driver class can specify the combiner.

The combiner() executes on each machine that performs a map task. Combiners optimize MapReduce task that locally aggregates before the shuffle and sort phase.

The same codes implement both the combiner and the reduce functions, combiner() on map node and reducer() on reducer node.

The main function of a Combiner is to consolidate the map output records with the same key.

The output (key-value collection) of the combiner transfers over the network to the Reducer task as input.

This limits the volume of data transfer between map and reduce tasks, and thus reduces the cost of data transfer across the network. Combiners use grouping by key for carrying out this function.

**The combiner works as follows**:

- ✓ It does not have its own interface and it must implement the interface at reduce().
- ✓ It operates on each map output key. It must have the same input andoutput key-value types as the Reducer class.
- ✓ It can produce summary information from a large dataset because itreplaces the original Map output with fewer records or smaller records.

## Reduced Tasks

Java API at Hadoop includes Reducer class. An abstract function, reduce() is in the Reducer.

- ✓ Any specific Reducer implementation should be subclass of this class and override the abstract reduce().
- ✓ Reduce task implements reduce() that takes the Mapper output (which shuffles and sorts), which is grouped by key-values (k2, v2) and applies it in parallel to each group.
- ✓ Intermediate pairs are at input of each Reducer in order after sorting using the key.

- ✓ Reduce function iterates over the list of values associated with a key and produces outputs such as aggregations and statistics.
- ✓ The reduce function sends output zero or another set of key-value pairs (k3, v3) to the final the output file. Reduce: {(k2, list (v2) -> list (k3, v3)}

## Sample code for Reducer Class

**public class ExarrpleReducer extends Reducer<k2, v2, k3, v3>**

**void reduce (k2 key, Iterable<V2> values, Context context) throws IOBxception, InterruptedBxception**

**{ ... }**

## Details of Map Reduce processing Steps.

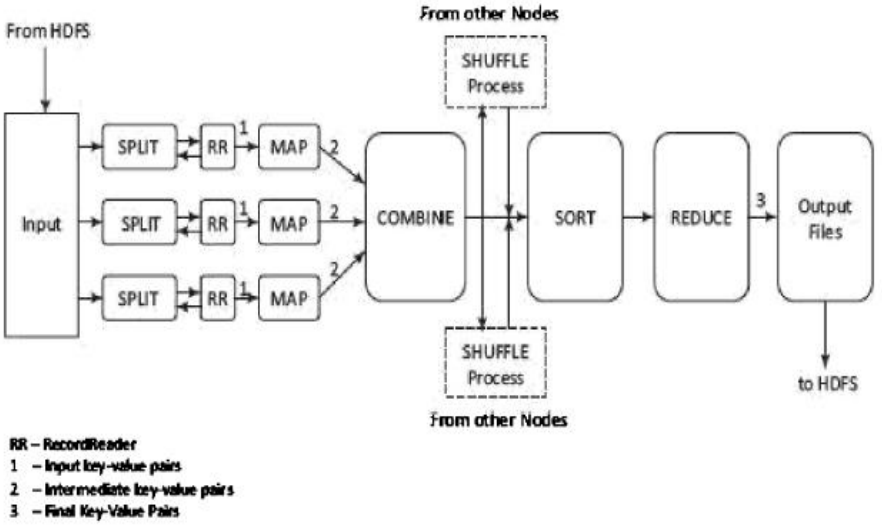



**Figure 4.6** MapReduce execution steps

Execution of MapReduce job does not consider how the distributed processing implements. Rather, the execution involves the formatting (transforming) of data at each step

Figure 4.6 shows the execution steps, data flow, splitting, partitioning and sorting on a map node and reducer on reducer node.

## Copying with Node Failure

The primary way using which Hadoop achieves fault tolerance is through restarting the tasks.

- ✓ Each task nodes (TaskTracker) regularly communicates with the master node, JobTracker. If a TaskTracker fails to communicate with the JobTracker for a pre-defined period (by default, it is set to 10 minutes), a task node failure by the JobTracker is assumed.
- ✓ The JobTracker knows which map and reduce tasks were assigned to each TaskTracker.
- ✓ If the job is currently in the mapping phase, then another TaskTracker will be assigned to re-execute all map tasks previously run by the failed TaskTracker.
- ✓ If the job is in the reducing phase, then another TaskTracker will re-execute all reduce tasks that were in progress on the failed TaskTracker.
- ✓ Once reduce tasks are completed, the output writes back to the HDFS. Thus, if a TaskTracker has already completed nine out of ten reduce tasks assigned to it, only the tenth task must execute at a different node.

The failure of JobTracker (if only one master node) can bring the entire process down; Master handles other failures, and the MapReduce job eventually completes. When the Master compute-node at which the JobTracker is executing fails, then the entire

MapReduce job must restart. Following points summarize the coping mechanism with distinct Node Failures:

- ✓ Map TaskTracker failure:
  - Map tasks completed or in-progress at TaskTracker, are reset to idle on failure
  - Reduce TaskTracker gets a notice when a task is rescheduled on another TaskTracker
- ✓ Reduce TaskTracker failure:
  - Only in-progress tasks are reset to idle
- ✓ Master JobTracker failure:
  - Map-Reduce task aborts and notifies the client (in case of one master node).

## COMPOSING MAPREDUCE FOR CALCULATIONS ANDALGORITHMS

MapReduce program composition in counting and summing, algorithms for relational algebraic operations, projections, unions, intersections, natural joins, grouping and aggregation, matrix multiplicationand other computations.

### Composing Map-Reduce for Calculations

The calculations for various operations compose are:

### Counting and Summing

- ✓ The number of alerts or messages generated during a specific maintenance activity of vehicles need counting for a month.
- ✓ From Figure 4.8 showed the pseudocode using emit() in the map() of *Mapper* class. *Mapper* emits 1 for each message generated.
- ✓ The reducer goes through thelist of ls and sums them. Counting is used in the data querying application.

- ✓ For example, count of messages generated, word count in a file, number of cars sold, and analysis of the logs, such as number of tweets per month. Application is alsoin business analytics field.

## Sorting

- ✓ From figure 4.6 illustrated MapReduce execution steps, i.e., dataflow, splitting, partitioning and sorting on a map node and reduce on a reducer node.
- ✓ *Mappers* just emit all items as values associated with the sorting keys which assemble as a function of items.
- ✓ *Reducers* combine all emitted parts into a final list.

## Finding Distinct Values (Counting unique values)

Applications such as web log analysis need counting of unique users.

Evaluation is performed for the totalnumber of unique values in each field for each set of records that belongs to the same group.

Two solutions are possible:

- ✓ The *Mapper* emits the dummy counters for each pair of field and groupld,and the *Reducer* calculates the total number of occurrences for each suchpair.
- ✓ The *Mapper* emits the values and groupld, and the *Reducer* excludes the duplicates from the list of groups for each value and increments the counter for each group.
- ✓ The final step is to sum all the counters emitted at the *Reducer*. This requires only one MapReduce job but the process is not scalable, and hence has limited applicability in large data sets.

## Collating

- ✓ Collating is a way to collect all items which have the same value of function in one document or file, or a way to process items with the same value of the function together.

- ✓ Examples of applications are producing inverted indexes and extract, transform and load operations.
- ✓ *Mapper* computes a given function for each item, produces value of the function as a key, and the item itself as a value.
- ✓ *Reducer* then obtains all item values using group-by function, processes or saves them into a list and outputs to the application task or saves them.

## Filtering or Parsing

- ✓ Filtering or parsing collects only those items which satisfy some condition or transform each item into some other representation.
- ✓ Filtering/parsing include tasks such as text parsing, value extraction and conversion from one format to another.
- ✓ Examples of applications of filtering arefound in data validation, log analysis and querying of datasets.
- ✓ *Mapper* takes items one by one and accepts only those items which satisfy the conditions and emit the accepted items or their transformed versions.
- ✓ *Reducer* obtains all the emitted items, saves them into a list and outputs to the application.

## Distributed Tasks Execution

- ✓ Large computations divide into multiple partitions and combine the results from all partitions for the final result.
- ✓ Examples of distributed running of tasks are physical and engineering simulations, numerical analysis and performance testing.
- ✓ *Mapper* takes a specification as input data, performs corresponding computations and emits results. *Reducer* combines all emitted parts into the final result.

## Graph Processing using Iterative Message Passing

- ✓ Graph is a network of entities and relationships between them. A node corresponds to an entity. An edge joining two nodes corresponds to a relationship.

- ✓ Path traversal method processes a graph. Traversal from one node to the next generates a result which passes as a message to the next traversal between the two nodes. Cyclic path traversal uses iterative message passing.
- ✓ A set of nodes stores the data and codes at a network. Each node contains a listof neighboring node IDs. MapReduce jobs execute iteratively. Each node in an iteration sends messages to its neighbors.
- ✓ Each neighbor updates its state based on the received messages. Iterations terminate on some conditions, such as completion of fixed maximal number of iterations or specified time to live or negligible changes in states between two consecutive iterations.
- ✓ *Mapper* emits the messages for each node using the ID of the adjacent node as a key. All messages thus group by the incoming node. *Reducer* computes the stateagain and rewrites a node new state.

## Cross Correlation

Cross-correlation involves calculation using number of tuples where the items co-occur in a set of tuples of items. If the total number of items is N, then the total number of values= N x N. Cross correlation is used in text analytics. (Assume that items are words and tuples are sentences). Another application is in market-analysis (for example, to enumerate, the customers whobuy item x tend to also buy y).

If N x N is a small number, such that the matrix can fit in the memory of a single machine, then implementation is straightforward.

Two solutions for finding cross correlations are:

- ✓ The *Mapper* emits all pairs and dummy counters, and the *Reducer* sums these counters.
- ✓ The benefit from using combiners is little, as it is likely that all pairs are distinct.

The accumulation does not use in-memory computations as N is very large.

- ✓ The *Mapper* groups the data by the first item in each pair and maintains an associative array ("stripe") where counters for all adjacent items accumulate.
- ✓ The *Reducer* receives all stripes for the leading item, mergesthem and emits the same result as in the pairs approach.

**The grouping**:

- ✓ Generates fewer intermediate keys. Hence, the framework has less sortingtodo.
- ✓ Greatly benefits from the use of combiners.
- ✓ In-memory accumulation possible.
- ✓ Enables complex implementations.
- ✓ Results in general, faster computations using stripes than "pairs".

## Matrix-Vector Multiplication by MapReduce

Numbers of applications need multiplication of n x n matrix A with vector **B** of dimension **n.** Each element of the product is the element of vector **C** of dimension **n.** The elements of C calculate by relation,

$c_i = \sum_{j=1}^{n} a_{ij} b_j$ . An example of calculations is given below.

Assume $\mathbf{A} = \begin{vmatrix} 1 & 5 & 4 \\ 2 & 1 & 3 \\ 4 & 2 & 1 \end{vmatrix}$ and $\mathbf{B} = \begin{vmatrix} 4 \\ 1 \\ 3 \end{vmatrix}$.

Multiplication $\mathbf{C} = \mathbf{A} \times \mathbf{B} = \begin{bmatrix} 1 \times 4 + 5 \times 1 + 4 \times 3 \\ 2 \times 4 + 1 \times 1 + 3 \times 3 \\ 4 \times 4 + 2 \times 1 + 1 \times 3 \end{bmatrix}$

Hence, $\mathbf{C} = \begin{bmatrix} 21 \\ 18 \\ 21 \end{bmatrix}$

*Algorithm for using MapReduce:* The Mapper operates on A and emits row-wise multiplication of each matrix element and vector element (aij x bj 'V i). The Reducer executes sum() for summing all values associated with each i and emits the element ci. Application of the algorithm is found in linear transformation.

## Relational – Algebra Operations

### Selection

Consider the attributenames (ACVM_ID, Date, chocolate_flavour, daily_sales). Consider relation

*R* = {(524, 12122017, KitKat, 82), (524, 12122017, Oreo, 72), (525 2122017, KitKat, 82), (525, 12122017, Oreo, 72), (526, 12122017, KitKat, 82), (526, 12122017, Oreo, 72)}.

Selection *ACVM_ID <= 525* (R) selects the subset *R*= {(524, 12122017, KitKat, 82), (524, 12122017, Oreo, 72), (525, 12122017, KitKat, 82), (525, 12122017, Oreo, 72)}.

Selection *chocolate_flavour= Oreo* selects the subset {(524, 12122017, Oreo, 72), (525, 12122017, Oreo, 72), (526, 12122017, Oreo, 72)}.

The *Mapper* calls test() for each tuple in a row. When test satisfies the selection criterion then emits the tuple.

The *Reducer* transfers the received input tuple as the output.

## Projection

Consider attribute names (ACVM_ID, Date, chocolate_flavour, daily_sales).

Consider relation R = {(524, 12122017, KitKat, 82), (524, 12122017, Oreo, 72)}. Projection II *ACVM_m (R)* selects the subset {(524)}.

Projection, II *chocolate_flavour, o.s* daily_sales* selects the subset {(KitKat, 0.5x82), (Oreo, 0.5 X 72)}

The *Mapper* calls test() for each tuple in a row. When the test satisfies, the predicate then emits the tuple (same as in selection).

The *Reducer* transfers the received input tuples after eliminating the possible duplicates. Such operations are used in analytics

## Union

Consider,

Rl = {(524, 12122017, KitKat, 82), (524, 12122017, Oreo, 72)}

R2 = {(525, 12122017, KitKat, 82), (525, 12122017, Oreo, 72)} and

R3 = {(526, 12122017, KitKat, 82), (526, 12122017, Oreo, 72)}

Result of Union operation between Rl and R3 is:

Rl U R3 = {(524, 12122017, KitKat, 82), (524, 12122017, Oreo, 72), (526, 12122017, KitKat, 82), (526, 12122017, Oreo, 72)}

The *Mapper* executes all tuples of two sets for union and emits all the resultant tuples.

The *Reducer* class object transfers the received input tuples after eliminating the possible duplicates.

## Intersection

Consider, Rl = {(524, 12122017, Oreo, 72)}

R2 = {(525, 12122017, KitKat, 82)}

and R3 = {(526, 12122017, KitKat, 82), (526, 12122017, Oreo, 72)}

Result of Intersection operation between Rl and R3 are

Rl n R3 = {(12122011, Oreo)}

The *Mapper* executes all tuples of two sets for intersection and emits all the resultant tuples.

The *Reducer* transfers only tuples that occurred twice. This is possible only when tuple includes primary key and can occur once in a set. Thus, both the sets contain this tuple.

## Difference

Consider:

**Rl** = {(12122017, **KitKat, 82),** (12122017, **Oreo,** 72)} and

R3 = {(12122017, KitKat, 82), (12122017, Oreo, 25)}

Difference means the tuple elements are not present in the second relation. Therefore, difference

set_l is Rl - R3 = (12122017, Oreo, 72) and

set_2 is R3 - Rl = (12122017, Oreo, 25).

The *Mapper* emits all the tuples and tag. A tag is the name of the set (say, set_l or set_2 to which a tuple belongs to).

The *Reducer* transfers only tuples that belong to set_l.

## Symmetric Difference

Symmetric difference (notation is *A fl. B* (or *A* e B)] is another relational entity. It means the set of elements in exactly one of the two relations A or B. R3 e Rl =

(12122017, Oreo, 25).

The *Mapper* emits all the tuples and tag. A tag is the name of the set (say, set_l or set_2 this tuple belongs to).

The *Reducer* transfers only tuples that belong to neither set_l or set_2.

## Natural Join

Consider two relations Rl and R2 for tuples a, band c. Natural Join computes for Rl (a, b) with R2 (b, c). Natural Join is R (a, b, c).

Tuples b joins as one in a Natural Join. The *Mapper* emits the key-value pair (b, (Rl, a)) for each tuple (a, b) of Rl, similarly emits (b, (R2, c)) for each tuple (b, c) of R2.

The *Mapper* is mapping both with Key for b. The *Reducer* transfers all pairs consisting of one with first component Rl and the other with first component R2, say (Rl, a) and (R2, c).

The output from the key and value list is a sequence of key-value pairs. The key is of no use and is irrelevant. Each value is one of the triples (a, b, c) such that (Rl, a) and (R2, c) are present in the input list of values.

## Grouping and Aggregation by MapReduce

Grouping means operation on the tuples by the value of some of their attributes after applying the aggregate function independently to each attribute. A Grouping operation denotes by <grouping attributes> j <function-list> (R). Aggregate functions are count(), sum(), avg(), min() and max().

Assume *R*= {(524, 12122017, KitKat, 82), (524, 12122017, Oreo, 72), (525,

12122017, KitKat, 82), (525, 12122017, Oreo, 72), (526, 12122017, KitKat, 82), (526, 12122017, Oreo, 72)}.

Chocolate_flavour i count ACVM_ID, sum (daily_sales (chocolate_flavour))

will give the output (524, KitKat, sale_month), (525, KitKat, sale_month), .... and (524, Oreo, sale_month), (525, Oreo, sale_month), .... for all **ACVM_IDs.**

The *Mapper* finds the values from each tuple for grouping and aggregates them. The *Reducer* receives the already grouped values in input for aggregation.

## Matrix Multiplication

Consider matrices named A (i rows andj columns) and B( rows and k columns) to produce the matrix C;(i rows and k columns). Consider the elements of matrices **A, B** and C as follows:

$$A = \begin{matrix} a_{11}\ a_{12}\ .\ .\ a_{1j} \\ a_{21}\ a_{22}\ .\ .\ a_{2j} \\ .\ .\ .\ .\ . \\ .\ .\ .\ .\ . \\ a_{i1}\ a_{i2}\ .\ .\ a_{ij} \end{matrix} \quad B = \begin{matrix} b_{11}\ b_{12}\ .\ b_{1k} \\ b_{21}\ b_{22}\ .\ b_{2k} \\ .\ .\ .\ . \\ .\ .\ .\ . \\ b_{j1}\ b_{i2}\ .\ b_{jk} \end{matrix} \quad C = \begin{matrix} c_{11}\ c_{12}\ .\ c_{1k} \\ c_{21}\ c_{22}\ .\ c_{2k} \\ .\ .\ .\ . \\ .\ .\ .\ . \\ c_{i1}\ c_{i2}\ .\ c_{ik} \end{matrix}$$

A.B = C; Each element evaluates as follow:

$$C_{ik} = \text{Sum}\ (a_{ij} \times b_{jk})\ j = 1\ \text{to}\ j.\ v_a = a_{ij}\ \text{and}\ v_b = b_{jk}$$

First Row of C

- ✓ C first column element= $(a_{11}b_{11} + a_{12}b_{21} + .... + a_{1j}b_{j1})$.
- ✓ Second column element= $(a_{11}b_{12} + a_{12}b_{22} + \quad + a_{1j}b_{jz})$,
- ✓ The $k^{th}$ column element= $(a_{11}b_{1} + a_{12}b_{2} + .... + a_{j1}\ b_{jk})$.

**Second row of** C

- ✓ C first column element = $(a_{21}b_{11} + a_{22}b_{21} + .... + a_{j2}\ b_{j1})$.

- ✓ Second column element = (a21b12+ a22b22+ + azj•bjz),
- ✓ The k[th] column element= ($a_{21}b_{1}$ + $a_{22}b_{2}$ + + azj· bjk).

## The i[th] row of C

- ✓ C first column element = (ailb$_{11}$+ a $_{2}$b$_{21}$ +.... + a ij.b $_{1}$).
- ✓ Second column element = (ailb$_{12}$+ a i$_{2}$b$_{22}$+.... + a ij.b $_{2}$).
- ✓ The k[th] column element= ($a_{i}b_{1}$ + a $_{2}$b$_{2}$ + ……..+aij.bjk).

## HIVE

Hive was created by Facebook. Hive is a data warehousing tool and is also a data store on the top of Hadoop. An enterprise uses a data warehouse as large data repositories that are designed to enable the tracking, managing, and analyzing the data.

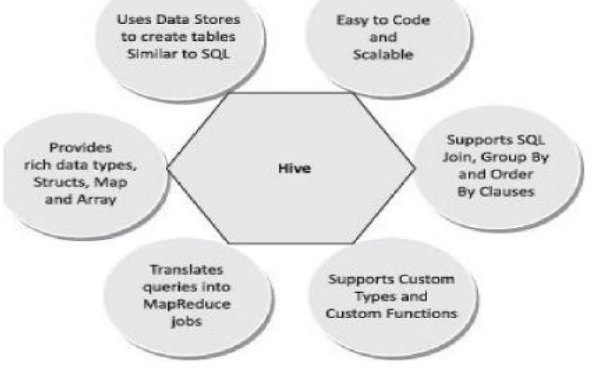



**Figure 4.9** Main features of Hive

## Hive Characteristics

- ✓ Has the capability to translate queries into MapReduce jobs. This makes Hive scalable, able to handle data warehouse applications, and therefore, suitable for the analysis of static data of an extremely large size, where the fast response-time is not a criterion.
- ✓ Supports web interfaces as well. Application APIs as well as web-browserclients, can access the Hive DB server.
- ✓ Provides an SQL dialect (Hive Query Language, abbreviated HiveQL or HQL).

*Results of HiveQL Query and the data load in the tables which store at the Hadoop cluster at HDFS.*

## Limitations of Hive is:

- ✓ Not a full database. Main disadvantage is that Hive does not provide update, alter and deletion of records in the database.
- ✓ Not developed for unstructured data.
- ✓ Not designed for real-time queries.
- ✓ Performs the partition always from the last column.

## HIVE ARCHITECTURE

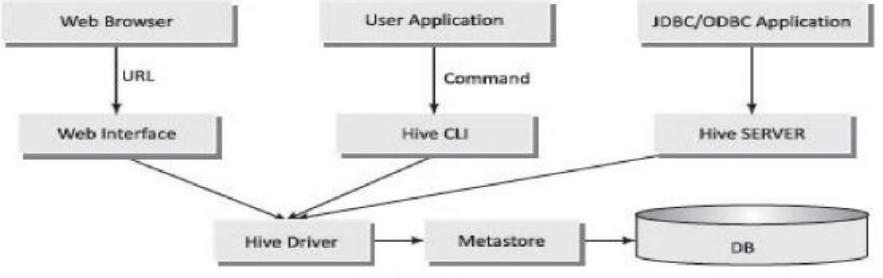



**Figure 4.10** Hive architecture

Components of Hive architecture are:

- ✓ **Hive Server (Thrift)** - An optional service that allows a remote client to submit requests to Hive and retrieve results. Requests can use a variety of programming languages.
- ✓ Thrift Server exposes a very simple client API toexecute HiveQL statements.
- ✓ **Hive CLI (Command Line Interface)** - Popular interface to interact with Hive. Hive runs in local mode that uses local storage when running the CLI on a Hadoop cluster instead of HDFS.
- ✓ **Web Interface** - Hive can be accessed using a web browser as well. This requires a HWI Server running on some designated code. The URL *http:// hadoop:<port no.>* / *hwi* command can be used to access Hive through the web.
- ✓ **Metastore** - It is the system catalog. All other components of Hive interact with the Metastore. It stores the schema or metadata of tables, databases, columns in a table, their data types and HDFS mapping.
- ✓ **Hive Driver** - It manages the life cycle of a HiveQL statement during compilation, optimization and execution.

## Comparison with RDBMS

Hive is a DB system which defines databases and tables. Hive analyzes structured data in DB. Hive has certain differences with RDBMS.

| Characteristics | Hive | RDBMS |
|---|---|---|
| Record level queries | No Update and Delete | Insert, Update and Delete |
| Transaction support | No | Yes |
| Latency | Minutes or more | In fractions of a second |
| Data size | Petabytes | Terabytes |
| Data per query | Petabytes | Gigabytes |

| | | |
|---|---|---|
| Query language | HiveQL | SQL |
| Support JDBC/ODBC | Limited | Full |

## Hive Data Types and File Formats

Hive defines various primitive, complex, string, date/time, collection data typesand file formats for handling and storing different data formats. The following Table gives primitive, string, date/time and complex Hive data types and their descriptions.

| Data TypeName | Description |
|---|---|
| **TINYINT** | 1 byte signed integer. Postfix letter is Y. |
| **SMALLINT** | 2 byte signed integer. Postfix letter is S. |
| **INT** | 4 byte signed integer |
| **BIGINT** | 8 byte signed integer. Postfix letter is L. |
| **FLOAT** | 4 byte single-precision floating-point number |
| **DOUBLE** | 8 byte double-precision floating-point number |
| **BOOLEAN** | True or False |
| **TIMESTAMP** | UNIX timestamp with optional nanosecond precision. It supports 0ava .sql.Timestamp format "YYYY-MM-DD HH:MM:SS.fffffffff' |
| **DATE** | YYYY-MM-DD format |
| **VARCHAR** | 1 to 65355 bytes. Use single quotes(') or double quotes("") |

| | |
|---|---|
| **CHAR** | 255 bytes |
| **DECIMAL** | Used for representing immutable arbitrary precision. DECIMAL (precision,scale) format |

The following Table gives Hive three Collection data types and their descriptions.

| File Format | Description |
|---|---|
| Text file | The default file format, and a line represents a record. The delimiting characters separate the lines. Text file examples are CSV, TSV,JSON and XML(Section 3.3.2). |
| Sequenti alfile | Flat file which stores binary key-value pairs, and supports compression. |
| RCFile | Record Columnar file (Section 3.3.3.3). |
| ORCFILE | ORC stands for Optimized Row Columnar which means it can store data in an optimized way than in the other file formats (Section 3.3.3.4). |

## HIVE Data Model

| Name | Description |
|---|---|
| Database | Namespace for tables |
| Tables | Similar to tables **in** RDBMS Support filter, projection, join and union operationsThe table data stores in a directory in HDFS |

| | |
|---|---|
| Partitions | Table can have one or more partition keys that tell how the data stores |
| Buckets | Data in each partition further divides into buckets based on hash of a columnin the table.Stored as a file in the partition directory. |

## Hive Integration and Workflow Steps

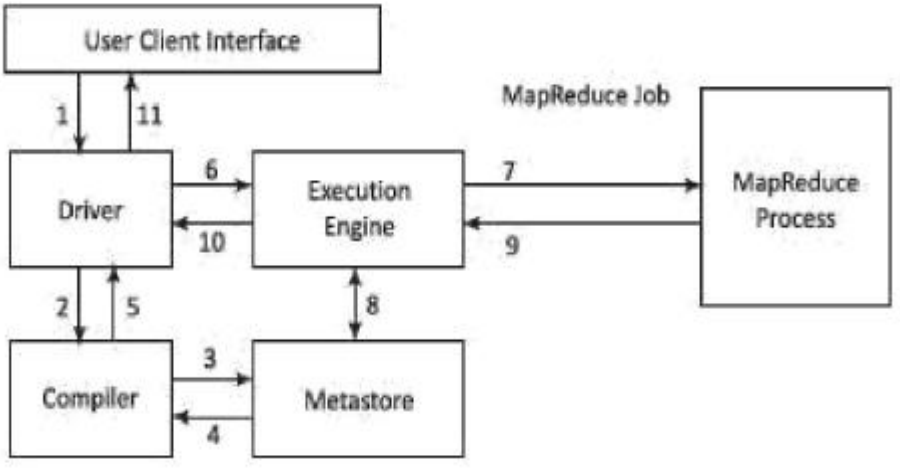



**Figure 4.11** Dataflow sequences and workflow steps

The workflow steps are as follows :

| |
|---|
| **Execute Query:** Hive interface (CLI or Web Interface) sends a query to DatabaseDriver to execute the query. |
| **Get Plan:** Driver sends the query to query compiler that parses the query to checkthe syntax and query plan or the requirement of the query. |

| |
|---|
| **Get Metadata:** Compiler sends metadata request to Metastore (of any database, such as MySQL). |
| **Send Metadata:** Metastore sends metadata as a response to compiler. |
| **Send Plan:** Compiler checks the requirement and resends the plan to driver. The parsing and compiling of the query is complete at this place. |
| **Execute Plan:** Driver sends the execute plan to execution engine. |
| **Execute Job:** Internally, the process of execution job is a MapReduce job. The execution engine sends the job to JobTracker, which is in Name node and it assigns this job to TaskTracker, which is in Data node. Then , the query executes the job. |
| **Metadata Operations:** Meanwhile the execution engine can execute the metadata operations with Metastore. |
| **Fetch Result:** Execution engine receives the results from Data nodes. |
| **Send Results:** Execution engine sends the result to Dr iver. |
| **Send Results:** Driver sends the results to Hive Interfaces. |

## Hive Built-in Functions

| **Return Type** | Syntax | Description |
|---|---|---|
| **BIGINT** | round(doublea) | Returns the rounded BIGINT (8 Byte integer) value of the 8 Byte double-precision floating point number a |
| **BIGINT** | floor(doublea) | Returns the maximum BIGINT value that is equal to or less thanthe double. |

| | | |
|---|---|---|
| **BIGINT** | ceil(double a) | Returns the minimum BIGINT value that is equal to or greater than the double. |
| double | rand(), rand(int seed) | Returns a random number (double) that distributes uniformly from O to 1 and that changes in each row. Integer seed ensured that random number sequence is deterministic. |
| string | concate(string strl, string str2, ...) | Returns the string resulting from concatenating strl with str2, ..... |
| string | substr(string str, int start) | Returns the substring of str starting from a start position till the end of string str. |
| string | substr(string str, int start,int length) | Returns the substring of str starting from the start position with the given length. |
| string | upper(string str), ucase (string str) | Returns the string resulting from converting all characters ofstr to upper case. |
| string | lower(string str), lcase(stringstr) | Returns the string resulting from converting all characters ofstr to lower case. |
| string | trim(stringstr) | Returns the string resulting from trimming spaces from both ends. trim ('12A34 56') returns '12A3456' |

| | | |
|---|---|---|
| string | ltrim(string str); rtrim(stringstr) | Returns the string resulting from trimming spaces (only one end, left or right hand side or right-handside spaces trimmed).<br>ltrim('12A34 56') returns '12A3456' and rtrim(' 12A34 56 ')returns '12A3456'. |
| string | rtrim(stringstr) | Returns the string resulting from trimming spaces from the end (right hand side) of str. |

| | | |
|---|---|---|
| int | year(string date) | Returns the year part of a date or a timestamp string. |
| int | month(string date) | Returns the month part of a date or a timestamp string . |
| int | day(string date) | Returns the day part of a date or a timestamp string. |

## HIVEQL

- ✓ Hive Query Language (abbreviated HiveQL) is for querying the large datasets which reside in the HDFS environment.
- ✓ HiveQL script commands enable data definition, data manipulation and query processing.
- ✓ HiveQL supports a large base of SQL users who are acquainted with SQL to extract information from data warehouses.

| | |
|---|---|
| HiveQL Process Engine | HiveQL is similar to SQL for querying on schema information at the Metastore. It is one of the replacements of traditional approach for MapReduce program . Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it. |
| Execution Engine | The bridge between HiveQL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results same as MapReduce results. It uses the flavor of MapReduce. |

### HiveQL Data Definition Language (DDL)

HiveQL database commands for data definition for DBs and Tables are CREATE DATABASE, SHOW DATABASE {list of all DBs), CREATE SCHEMA, CREATE TABLE.

Following are HiveQL commands which create a table:

CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [<database name>.]

<table name>

[(<column name> <data type> [COMMENT <column comment>], ...)] [COMMENT <table comment>]

[ROW FORMAT <row format>][STORED AS <file format>]

- ✓ A command is

CREATE DATABASEISCHEMA [IF NOT EXISTS] <database name>;

IF NOT EXISTS is an optional clause. The clause notifies the user that a database with the same name already exists. SCHEMA can be also created in place of DATABASE using this command

A command is written to get the list of all existing databases. SHOW DATABASES;

A command is written to delete an existing database.

DROP (DATABASEISCHEMA) [RESTRICT I CASCADE]; [IF EXISTS] <database name>

## HiveQL Data Manipulation Language (DML)

HiveQL commands for data manipulation are USE <database name>, DROP DATABASE, DROP SCHEMA, ALTER TABLE, DROP TABLE, and LOAD DATA.

The following is a command for inserting (loading) data into the Hive DBs.

LOAD DATA [LOCAL] INPATH '<file path>' [OVERWRITE] INTO

TABLE <table name> [PARTITION (partcoll=vall,partcol2=val2 ...)]

LOCAL is an identifier to specify the local path. It is optional. OVERWRITE is optional to overwrite the data in the table. PARTITION is optional. vall is value assigned to partition column 1 (partcoll) and val2 is value assigned to partition column 2 (partcol2).

## HiveQL For Querying the Data

Partitioning and storing are the requirements. A data warehouse should have a

large number of partitions where the tables, files and databases store. Querying then requires sorting, aggregating and joining functions.

Querying the data is to SELECT a specific entity *satisfying* a condition, *having* presence of an entity or selecting specific entity using GroupBy .

SELECT [ALL I DISTINCT] <select expression>, <selectexpression>, ...

FROM <table name>

[WHERE <where condition>] [GROUP BY <column List>] [HAVING <having condition>]

[CLUSTER BY <column List>I [DISTRIBUTE BY <column List>] [SORT BY <column List>]]

[LIMIT number];

## PIG

- ✓ It is an abstract over MapReduce
- ✓ It is an execution framework for parallel processing
- ✓ Reduces the complexities of writing a MapReduce program
- ✓ Is a high-level dataflow language. Dataflow language means that a Pig operation node takes the inputs and generates the output for the next node
- ✓ Is mostly used in HDFS environment
- ✓ Performs data manipulation operations at files at data nodes in Hadoop.

## Applications of Apache Pig

- ✓ Analyzing large datasets
- ✓ Executing tasks involving adhoc processing
- ✓ Processing large data sources such as web logs and streaming online data
- ✓ Data processing for search platforms. Pig processes different types of data
- ✓ Processing time sensitive data loads; data extracts and analyzes quickly.

Differences between Pig and MapReduce

| Pig | MapReduce |
|---|---|
| A dataflow language | A data processing paradigm |
| High level language and flexible | Low level language and rigid |
| Performing Join, filter, sorting or ordering operations are quite simple | Relatively difficult to perform Join, filter, sorting or ordering operations between datasets |
| Programmer with a basic knowledge of SQL can work conveniently | Complex Java implementations requireexposure to Java language |
| Uses multi-query approach, thereby reducing the length of the codes significantly | Require almost 20 times more the number of lines to perform the same task |
| No need for compilation for execution; operators convert internally into MapReduce jobs | Long compilation process for Jobs |
| Provides nested data types like tuples, bagsand maps | No such data types |

**Differences between Pig and SQL**

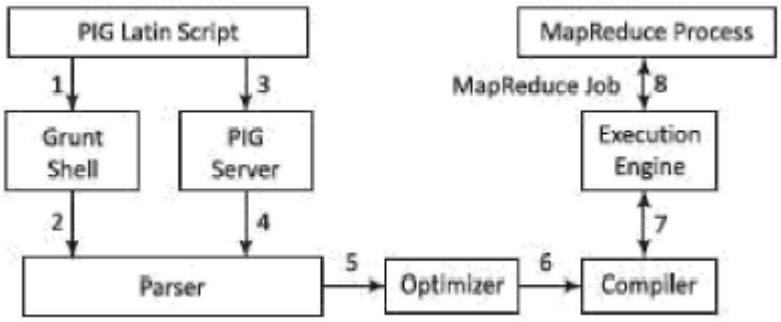
| Pig | SQL |
|---|---|
| Pig Latin is a procedural language | A declarative language |
| Schema is optional, stores data without assigning a schema | Schema is mandatory |
| Nested relational data model | Flat relational data model |
| Provides limited opportunity for Query optimization | More opportunity for query optimization |

Pig and Hive codes, both create MapReduce jobs when execute. Hive in some cases, operates on HDFS in a similar way Apache Pig does.

## Differences between Pig and Hive

| Pig | Hive |
|---|---|
| Originally created at Yahoo | Originally created at Facebook |
| Exploits Pig Latin language | Exploits HiveQL |
| Pig Latin is a dataflow language | HiveQL is a query processinglanguage |
| Pig Latin is a procedural language and it fits in pipelineparadigm | HiveQL is a declarative language |
| Handles structured, unstructured and semi-structureddata | Mostly used for structured data |

## Pig Architecture




The three ways to execute scripts are:

1. **Grunt Shell:** An interactive shell of Pig that executes the scripts.
2. **Script File:** Pig commands written in a script file that execute at Pig Server.
3. **Embedded Script:** Create UDFs for the functions unavailable in Pig built- in operators. UDF can be in other programming languages. The UDFs can embed in Pig Latin Script file.

**Parser** A parser handles Pig scripts after passing through Grunt or Pig Server. The Parser performs type checking and checks the script syntax. The output is a Directed Acyclic Graph (DAG).

Acylic means only one set of inputs are simultaneously at a node, and only one set of output generates after node operations.

DAG represents the Pig Latin statements and logical operators. Nodes represent the logical operators. Edges between sequentially traversed nodes represent the dataflows.

**Optimizer** The DAG is submitted to the logical optimizer. The optimization activities, such as split, merge, transform and reorder operators execute in this phase. The optimization is an automatic feature.

The optimizer reduces the amount of data in the pipeline at any instant of time, while processing the extracted data. It executes certain functions for carrying out this task, as explained as follows:

***PushUpFilter:*** If there are multiple conditions in the filter and the filter can be split, Pig splits the conditions and pushes up each condition separately. Selecting these conditions at an early stage helps in reducing the number of records remaining in the pipeline.

***PushDownf or EachFlatten:*** Applying flatten, which produces a cross product between a complex type such as a tuple, bag or other fields in the record, as late as possible in the plan. This keeps the number of records low in the pipeline.

***ColumnPruner:*** Omitts never used columns or the ones no longer needed, reducing the size of the record. This can be applied after each operator, so that the fields can be pruned as aggressively as possible.

**MapKeyPruner**: Omitts never used map keys, reducing the size of the record.

**Limit Optimizer***:* If the limit operator is immediately applied after *load* or *sort* operator, Pig converts the load or sort into a limit-sensitive implementation, which does not require processing the whole dataset. Applying the limit earlier reduces the number of records.

**Compiler** The compiler compiles after the optimization process. The optimized codes are a series of MapReduce jobs.

**Execution Engine** Finally, the MapReduce jobs submit for execution to the engine. The MapReduce jobs execute and it outputs the final result.

## Apache- Pig Grunt Shell

Main use of Grunt shell is for writing Pig Latin scripts. Any shell command invokes using sh and ls. Syntax of **sh** command is:

grunt> sh shell command parameters

Syntax of **ls** command:

grunt> sh ls

## Pig Latin Data Model

Pig Latin supports primitive data types which are atomic or scalar data types. Atomic data types are int, float, long, double, char[], byte [].

The language also defines complex data types. Complex data types are tuple, bag and map.

## Data types and examples

| Data type | Description | Example |
|---|---|---|
| bag | Collection of tuples | {(1,1), (2,4)} |
| tuple | Ordered set of fields | (1,1) |
| map (data map) | Set of key-value pairs | [Number#l] |
| int | Signed 32-bit integer | 10 |
| long | Signed 64-bit integer | lOL or 101 |
| float | 32-bit floating point | 22.7F or 22.7f |
| double | 64-bit floating point | 3.4 or 3.4e2 or 3.4E2 |
| chararray | Char [], Character array | data analytics |
| bytearray | BLOB (Byte array) | ffoo |

## Pig Latin and Developing Pig Latin scripts

Pig Latin enables developing the scripts for data analysis. A number of operators in Pig Latin help to develop their own functions for reading, writing and processing data. Pig Latin programs execute in the Pig run-time environment.

## Pig Latin

- ✓ Basic constructs to process the data.

- ✓ Include schemas and expressions.
- ✓ End with a semicolon.
- ✓ LOAD statement reads the data from file system, DUMP displays the resultand STORE stores the result.
- ✓ Single line comments begin with - - and multiline begin with/* and end with*/
- ✓ Keywords (for example, LOAD, STORE, DUMP) are not case-sensitive. Function names, relations and paths are case-sensitive.

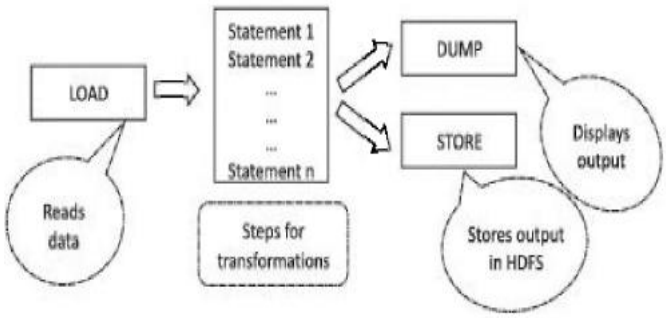



**Figure 4.15** Order of processing Pig statements—Load, dump, and store

## Apache Pig Execution

**Pig Execution Modes** Local Mode: All the data files install and run from a local host using the local file system. Local mode is mostly used for testing purpose.

**COMMAND:**

```
pig -x local
```

MapReduce Mode:

All the data files load or process that exists in the HDFS. A MapReduce job invokes in the back-end to perform a particular operation on the data that exists in the HDFS when a Pig Latin statement executes to process thedata.

**COMMAND**:

```
pig -x mapreduce or pig
```

## Pig Latin Script Execution Modes

- ✓ Interactive Mode - Using the Grunt shell.

- ✓ Batch Mode - Writing the Pig Latin script in a single file with .pig extension.
- ✓ Embedded Mode - Defining UDFs in programming languages such as Java, and using them in the script.

## Commands

- ✓ To get the list of pig commands: *pig-help;*
- ✓ To get the version of pig: *pig -version.*
- ✓ To start the Grunt shell, write the command: *pig*

**LOAD Command** The first step to a dataflow is to specify the input.

**Load statement in Pig Latin loads the data from** PigStorage.

**To load data from HBase:** book load 'MyBook' using HBaseStorage();

**For reading CSV file,** PigStorage **takes an argument which indicates which character to use as a separator.**

**For example,**

book = LOAD 'PigDemo/Data/Input/myBook.csv' USING PigStorage (,);

**To specify the data-schema for loading:** book = LOAD 'MyBook' AS (name, author, edition, publisher);

**Store Command** Pig provides the store statement for writing the processed **data after the processing is complete.** It **is the mirror image of the load statement in certain ways.**

**By default, Pig stores data on HDFS in a tab-delimited file using PigStorage:**

STORE processed into '/PigDemo/Data/Output/Processed';

**To store in HBaseStorage with a *using* clause:** STORE processed into 'processed' using HBaseStorage();

**To store data as comma-separated text data, PigStorage takes an argument to indicate which character to use as a separator:** STORE processed into 'processed' using PigStorage(',');

**Dump Command** Pig provides dump command to see the processed data on the screen. This is particularly useful during debugging and prototyping sessions. It can also be useful for quick adhoc jobs.

The following command directs the output of the Pig script on the display screen:

DUMP processed;

## Relational Operations

The relational operations provided at Pig Latin operate on data. They transform data using sorting, grouping,joining, projecting and filtering. Followings are the basic relational operators:

**Foreach** FOREACH gives a simple way to apply transformations based on columns. It is Pig's projection operator.