

### Basic System Design

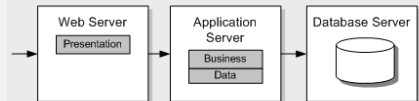
|                    |  |
|--------------------|--|
| Presentation Layer | User interface, caching, validation, single page or multi page |
| Business Layer     | Logic/workflows, reuse common logic                            |
| Data Layer         | Entity objects that pass data, database type. SQL vs NoSQL     |

Also consider security of application.

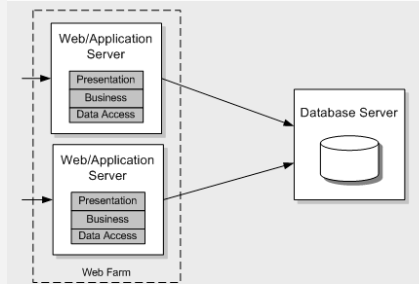
### System Design - Deployment Considerations



Non-distributed deployment of a Web application.



Distributed deployment of a Web application.

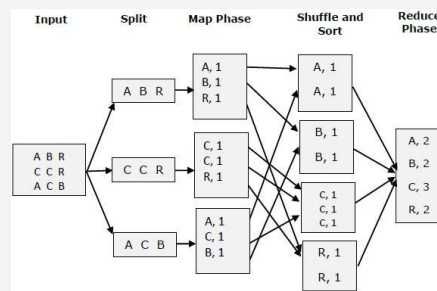


Load balancing a Web application.

Consider the following guidelines for deployment:

- Consider using non-distributed deployment to maximize performance.
- Consider using distributed deployment to achieve better scalability and to allow each layer to be secured separately.

### Map Reduce



The MapReduce algorithm contains two important tasks, namely Map and Reduce. The Map task takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value pairs). The Reduce task takes the output from the Map as an input and combines those data tuples (key-value pairs) into a smaller set of tuples.

### SQL vs NoSQL

**SQL** Language used for relational databases

Scaled vertically by increasing power (more common), scaled horizontally by partitioning

Tables and columns, rows, Have constrained logical relationships

Must exhibit ACID properties

MS-SQL, Oracle, Access, Ingress

**NoSQL** Language used for non relational dbs

Scales better horizontally using master-slave architecture

Multiple formats: Column, Key-Value, Document, Graph

### SQL vs NoSQL (cont)

Adheres to CAP

MongoDB, DynamoDB, CouchDB

Use data is small

SQL when:

Conceptually modeled as tabular

consistency is critical

Use Graph or hierarchial data

NoSQL when:

Data sets which are both large and mutate significantly

Businesses growing extremely fast but lacking data schemata

**ACID** - Atomicity, Consistency, Isolation, Durability

**CAP** - Consistency, Availability, Partition tolerance

### Algorithms

| Algorithm      | BEST | AVERAGE          | WORST            |
|----------------|------|------------------|------------------|
| Insertion Sort | n    | ¼ n <sup>2</sup> | ½ n <sup>2</sup> |

|            |          |        |        |
|------------|----------|--------|--------|
| Merge Sort | ½ n lg n | n lg n | n lg n |
|------------|----------|--------|--------|

|            |        |          |                  |
|------------|--------|----------|------------------|
| Quick Sort | n lg n | 2 n ln n | ½ n <sup>2</sup> |
|------------|--------|----------|------------------|

Quick Sort works better for small arrays  
Merge Sort works better for linked lists and is consistent for any size of data

### C# String Methods

**CompareTo()** Compare two strings  
`str2.CompareTo(str1)`

**IndexOf()** Returns the index position of first occurrence of character  
`str1.IndexOf(" ")`

**Remove()** deletes all the characters from beginning to specified index position.  
`str1.Remove(i);`

**Replace()** replaces the specified character with another  
`str1.Replace('old', 'new');`

**Substring()** this method returns substring.  
`str1.Substring(1, 7);`

`Substring(Int32)`  
`Substring(Int32, Int32) //start, length`

### C# List Methods

**BinarySearch()** Uses a binary search algorithm to locate a specific element in the sorted List<T> or a portion of it.

**ConvertAll(Converter)** Converts the elements in the current List<T> to another type, and returns a list containing the converted elements.

**IndexOf()** Returns the zero-based index of the first occurrence of a value in the List<T> or in a portion of it.

### C# List Methods (cont)

**Sort()** Sorts the elements or a portion of the elements in the List<T>

**Reverse()** Reverses the order of the elements in the List<T> or a portion of it.

Sort is QuickSort

### Search Basics

**Breadth First Search** An algorithm that searches a tree (or graph) by searching levels of the tree first, starting at the root.

Moves left to right on level, tracking children. Then moves to next level

**Depth First Search** An algorithm that searches a tree (or graph) by searching depth of the tree first, starting at the root.

It traverses left down a tree until it cannot go further

traverses back up trying the right child of nodes on that branch, and if possible left from the right children

### Search Basics (cont)

When finished examining a branch it moves to the node right of the root then tries to go left on all it's children until it reaches the bottom.

**When to use BFS:** Optimal for searching a tree that is wider than it is deep

Uses a queue to store information about the tree while it traverses a tree so uses more memory than DFS

**When to use DFS:** Optimal for searching a tree that is deeper than it is wide.

Uses a stack to push nodes onto.



By DesertGarnet

[cheatography.com/desertgarnet/](https://cheatography.com/desertgarnet/)

Not published yet.

Last updated 25th February, 2022.

Page 2 of 2.

Sponsored by [CrosswordCheats.com](https://crosswordcheats.com)

Learn to solve cryptic crosswords!

<http://crosswordcheats.com>