LIYANA ARACHCHIGE RANIL

ARCHITECTURE DECOMPOSITION

1. There are five ways that an Architect can decompose a large system in to smaller components

    a. Layering or Distribution
    b. Exposure , Functionality or Generality
    c. Coupling and Cohesion (Low Coupling , High Cohesion) or Volatility
    d. Configuration
    e. Planning , Tracking and Work Assignment

2. If a system is decomposed by Layering , it will not be decomposed by Distribution
3. The system would be decomposed until you see that it can be decomposed by any of the above mentioned in the same order
4. System will be decomposed in the order above mentioned
5. System will not be decomposed first by Planning, Tracking and Work Assignment and then trying to do Layering and Distribution. Instead follow the above order from “a” to “e”
6. **Layering** > Order the system in to different layers where by each layer provides some service to the upper layer while consuming some service from the layer underneath
7. Layering is some sort of an ordering of Principles or sort of ordering of abstractions
8. Layering is one of the top level decomposition and others would follow that
9. Layering can be done in Tier or in a Layer it self
10. **Distribution** > Distribution is performed among “**COMPUTATIONAL RESOURCES**”
11. Distribution could be resulted due to following nature of the large system ,
    a. Dedicated task own, their thread of control. Hence some processes do not need to wait for this process
    b. Many clients need access
    c. Greater fault isolation
    d. Distribution for separation of concern can be applied with redundancy which allows for high availability
12. **Exposure** > How a components is exposed and consumes other components. Any given component has three major task , **services** , **logic** and **integration**
13. **Functionality** > functionality is **grouping within the problem domain** and separating concerns based on that. For example

login module , customer module , inventory module etc. If the operational process is in mind , this is easilty achievable

14. **Generality** > decomposing based on generality means trying to understand components which can be used in other places as well. But when it comes to generality it is important not to make a component general for a requirement that does not exist
15. **Coupling & Cohesion** > this decomposition says keep things together that work together and keep things away that works away. So it is better to have low coupling and high cohesion
16. **Volatility** > Decomposing bases on volatility means identifying the sections or parts in the larger system which may **change frequently**. And keep those together. Usually this is applicable to **User Interfaces**.
17. **Configuration** > having a target system that must support different configurations. For example **security , performance , usability** etc
18. **Planning , Tracking** > Attempt to develop a fine grained project plan that takes into account **Ordering** dependencies and **Sizing**
    a. Ordering is understanding the dependencies among modules
    b. Sizing is dividing the tasks in to smaller pieces which can be developed in an **iterative** pattern
    c. A good architecture must have very limited number of bi-directional or circular dependencies
19. **Work Assignment** > Decomposing the system based on the skill set of the team members working on the project. This is the last thing to do when all above decomposition strategies have been applied

TIERS

1. Tier is a logical or physical organization of components in to an ordered chain of service providers and consumers
2. A tier would consume services from adjacent service provider tier and would provide services to adjacent service consumer tier
3. Tiers in tradition architectures are **CLIENT** , **WEB (PRESENTATION)** , **BUSINESS** , **INTEGRATION (MIDDLEWARE)** , **RESOURCE**
4. **CLIENT TIER** > any device that is used to access the display and local interaction of a system can be considered as a client tier. This tier is very much not in control and highly volatile and varying in nature

5. **WEB (PRESENTATION) TIER** > the tier which deals with rendering , personalization of views , content formatting , content transformation etc is considered to be WEB or Presentation tier. **This tier manages channel specific user sessions**
6. **BUSINESS TIER** > any tier which extract away the business logic is the business tier. Usually PRESENTATION tier uses BUSINESS tier services to execute the processing logics. The Heart of a system can be considered to be lying in the business tier. Business tier services may include activities from sending mails, authentication, authorization services and also managing modules such as customer profile, pay role etc.. This tier manages **TRANSACTIONS**
7. **INTEGRATION TIER** > This tier would help integrating the business tier to external data sinks, data feeds, data bases etc. The tier is also known as **middleware** tier. Due to the **varied and external nature of this tier**, the tier itself is kept as very much loosely coupled. Loosely coupled paradigms used could be **QUEUEING**, PUBLISH/SUBSCRIBER COMMUNICATION. SYSNCHRONOUS AND ASYNCHRONOUS POINT TO POINT MESSAGING etc.
8. **RESOURCE TIER** > This tier is identified as the data sources, sinks, feeds etc. This tier is abstracted away as Integration tier to keep loosely coupled nature. This tier is also known as **DATA TIER**
9. **TIERS** represents processing chains across components while **LAYER(S)** represents container / component relationships

LAYERS

1. A layer could be separation of principles or some abstractions
2. A layer is a hardware and software stack that **host some services within** a given **TIER**
3. Layers have well defined ordered relationship and interface boundaries
4. Layers represents container , component relationships
5. Typical layers are **Application** , **Virtual Platform** (Component API) , **Application Infrastructure** (Containers) , **Enterprise Services** (OS and Virtualization) , **Compute and Storage** (Hardware) , **Networking** (Switcher , Routers etc)
6. **APPLICATION LAYER** > after delegating all the responsibilities to other layers whatever is left are application specific.

Application layer is what makes a system unique from other systems. Combines user and business functionality

7. **VIRTUAL PLATFORM (Component API)** > Virtual Platform consist of the Component APIs on which APPLICATION is developed. Ex. Servlet API , JDBC api , EJB API etc.
8. **APPLICATION INFRASTRUCTURE (Container)** > Application infrastructure is provided by the containers it self. Ex, Tomcat , EJB container etc
9. **ENTERPRISE SERVICES (OS and VIRTUALIZATION)** > This layer provides necessary OS and virtualization support for Application infrastructure
10. **COMPUTE and STORAGE (Hardware layer)** > Contains the physical hardware. Enterprise services run on the compute and hardware layer
11. **NETWORKING INFRASTRUCTURE** > anything to do with networking, load balancers, routers etc.

SERVICE LEVEL REQUIREMENTS

1. Service Level Requirements are also known as Non Functional Requirements or QoS (Quality of Service) as well
2. Service level requirements that must be address by a typical architecture are **SCALABILITY** , **RELIABILITY** , **AVAILABILITY** , **EXTENSIBILITY** , **MAINTAINABILITY**, **MANAGEABILITY** , **PERFORMANCE** , **SECURITY** (All in all 8)
3. As an architect you must be able to achieve the stakeholder expectation between these quality of service requirements
4. If the main interest is PERFORMANCE , then you have to sacrifice the EXTENSIBILITY and MANAGEABILITY of the system may be
5. **PERFORMANCE** > time taken for a given screen transaction per user (In terms of RESPONSE TIME) or number of transactions for a given time , usually 1 sec (TRANSACTION THROUGHPUT)
6. **SCALABILITY** > As the load increases system should be able to provide the expected Quality Of Service without any changes to the system.

    a. A system can be though of as scalable only if it as the load increases the system still responds within acceptable limits
    b. Capacity is the maximum number of processes that a system can sustain under the existing infrastructure while still maintaining the quality of service

c. If a system is running at it's capacity and if it can not respond within allowable limits the system has reached it's **maximum scalability**
d. To scale a system that has met with its capacity **you MUST add additional HARDWARE**
e. This additional hardware can be added **HORIZONTALLY** or **VERTICALLY**
f. **Horizontal scaling** is adding new servers while Vertical scaling is adding new RAM and processors , disk to the **CURRENT machine**
g. **Horizontal Scaling** is adding more machines to the environment hence increasing the overall capacity of the system
h. **Horizontal Scaling is difficult than Vertical Scaling**, making a system work like one which is deployed in few other machines is much more difficult

7. **RELIABILITY** > As the load on the system increases system should function as the way it functioned before the increased load
    a. **Reliability ensure the INTERGRITY and CONSISTANCY of the application and all of its TRANSACTIONS**
    b. If a system is not RELIABLE the system is not SCALABLE. Hence the RELIABILITY aspect has a NEGATIVE impact on SCALABILITY
    c. For a system to truly SCALE , it MUST be RELIABLE

8. **AVAILABILITY** > A system must be accessible by users at all the times.
    a. **RELIABILITY can CONTRIBUTE to AVAILABILITY (RELIABILITY -> AVAILABILITY)**
    b. **But AVAILABILITY can be achieved even if a component fails (AVAILABILITY ->X , RELIABILITY)**
    c. By setting up an environment of redundant components and failover AVAILABILITY can be achieved

9. **EXTENSIBILITY** > extensibility is ability to add new feature to the system without effecting the existing functionality.
    a. You can not measure EXTENSIBILITY at the system deployment time
    b. It shows up at the first time when you try to add a new functionality

   c. To have a system support extensibility try **"LOW COUPLING" , "HIGH COHESION" , "PROGRAMMING FOR INTERFACES" , "ENCAPSULATION"**

10. **MAINTAINABILITY** > How easy to maintain the system, how easy to fix errors or bugs in the system with out effecting the existing system. Maintainability would be improved if there is good **DOCUMENTATION , MODULARIZATION ,LOW COUPLING** etc
    a. This system quality can not be measured at the deployment time
    b. Only when system flaws need to be corrected , this would come up

11. **MANAGEABILITY** > Manageability is the ability to manage the system to ensure the continued health of the system with respect to **scalability, availability, reliability, security, performance**.

    a. Manageability deals with system monitoring of the **QoS** requirements and ability to change the system configuration to improve the QoS dynamically without changing the system.
    b. Your architecture must have the ability to **MONITOR** the system and allow for **DYNAMIC** system configuration
12. **SECURITY** > security is the ability to ensure that the system is **NOT COMPROMISED**
    a. Creating an architecture that is separated into functional components makes it easy to secure the system since you can build **security zones** around the components
    b. If a component is failed , you can contain the security violation to that component

DIMENTIONS OF SYSTEMS

1. From a system computational point of view , you can think of the layout of an architecture (Tiers and Layers) as having six independent variables that are expressed as dimensions
   a. CAPACITY
   b. REDUNDANCY
   c. MODULARITY
   d. TOLERANCE
   e. WORKLOAD

   f. HETEROGENEITY
      (CRM to Work heta)

2. **CAPACITY** > Capacity dimension is the **RAW power** in an ELEMENT.

   a. CAPACITY is increased through VERTICAL SCALING
   b. VERTICAL SCALING is also know as HEIGHT
   c. CAPACITY can IMPROVE , PERFORMANCE , AVAILABILITY , SCALABILITY
3. **REDUNDANCY** > Multiple systems that work on the same job
   a. Load balancing among several web servers is an example
   b. REDUNDANCY is increased through HOROZONTAL SCALING
   c. HORIZONTAL SCALING is known as WIDTH
   d. REDUNDANCY can increase PERFORMANCE , RELIABILITY , AVAILABILITY, EXTENSIBILITY,SCALABILITY
   e. REDUNDANCY can decrease PERFORMANCE,MANGEABILITY and SECURITY
4. **MODULARITY** > how you divide a computational problem in to separate elements and spread those elements across multiple computer systems
   a. Modularity indicates how far into a system you have to go to get the data you need
   b. MODULARITY can increase SCALABILITY , EXTENSIBILITY , MAINTAINABILITY , SECURITY
   c. MODULARITY can decrease PERFORMANCE,RELIABILITY,AVAILABILITY and MANAGEABILITY
5. **TOLERANCE** > The time available to full fill a request from a user
   a. TOLERANCE is closely bound with the overall perceived PERFORMANCE
   b. TOLERANCE can increase PERFORMANCE , SCALABILITY , RELIABILITY , MANAGEABILITY
6. **WORKLOAD** > Computational work being performed at a particular point within the system
   a. WORKLOAD is closely related to CAPACITY in that workload consumes available CAPACITY
   b. WORKLOAD can increase PERFORMANCE , SCALABILITY , AVAILABILITY
7. **HETEROGENEITY** > Diversity in technologies that is used within a system or one of its subsystems
   a. HETEROGENETY comes from the variation of technologies that are used within the system

b. This could happen due to gradual accumulation over time , inheritance or acquisition
c. HETEROGENITY can increase PERFORMANCE , SCALABILITY
d. HETEROGENETY can decrease PERFORMANCE , SCALABILITY , AVAILABILITY,EXTENSIBILITY , MANAGEABILITY and SECURITY

INCREASED	Scalability	Availability	Reliability	Extensibility	Manageability	Maintainability	Performance	security
Capacity	increased	increased					increased	
Redundancy	increased	increased	increased	increased	decreased		increased/ decrease	decreased
Modularity	increased	decreased	decreased	increased	decreased	increased	decreased	increased
Tolerence	increased		increased		increased		increased	
Workload	increased	increased					increased	
Heterogeneity	increased/ decrease	decreased		decreased	decreased		increased/ decrease	decreased

TIERS	Security	Availability	Scalability	Extensibility	Manageability	Maintainability	Reliability	Performance
TWO TIER	Advantages	Disadvantages	Disadvantages	Disadvantages	Disadvantages	Disadvantages	Non Of Two	Advantages
THREE TIER	Non Of Two	Advantages	Advantages	Advantages	Advantages	Advantages		Any of Two
N-TIER	Non Of Two	Advantages	Advantages	Advantages	Advantages	Advantages		Any of Two

COMMON PRACTICES FOR IMPROVING SERVICE LEVEL REQUIREMENTS

1. INTRODUCING REDUNDANCY TO THE SYSTEM ARCHITECTURE

   The choice depends on the COST of the implementation and the REQUIREMENT. (Such as PERFORMANCE and SCALABILITY)

2. LOAD BALANCING

   a. Implement load balancing to address architectural concerns such as THROUGHPUT and SCALABILITY
   b. Load balancing helps you distribute the workload across several smaller machines instead of using one large machine
   c. This typically results in lower COST and better use of computing resources
   d. To implement load balancing , you usually select a load-balancer implementation based on its PERFORMANCE and AVAILABILITY

      i. Load balancing in network switches **> advantage of speed**
      ii. Load balancers in cluster management software and application servers **> managed closer to the application components , which gives greater flexibility and manageability**
      iii. Load Balancers based on the server instance DNS configuration > **Advantage of being simple to set up, does not address the session affinity**

   e. Load balancing provides variety of ALGORITHMS

      i. **Round Robin ALG** – Picks each server in turn
      ii. **Response-time or first available ALG** – Constantly monitor the response time of the server and pick the one which responds quickly
      iii. **Least loaded ALG** – Constantly monitor server load and select the server that has the most available capacity
      iv. **Weighted ALG** – specifies a priority on the preceding algorithms, giving some servers more work load than others
      v. **Client DNS-based ALG** – Distribute the load based on the clients DNS host and domain name info.

3. FAILOVER

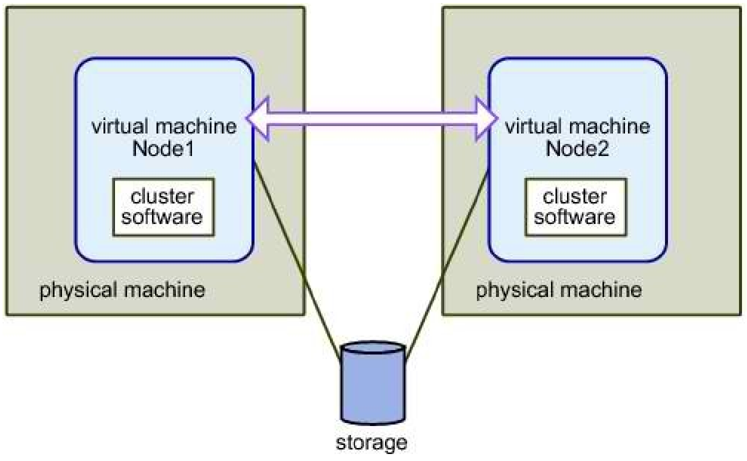
    a. You can use to minimize the likelihood of SYSTEM FAILURE
    b. Failover is a system configuration which allows one server to assume identity of a failing server within the network
    c. One important aspect of failover is available CAPACITY, which can be handled in two ways
        i. Designing with extra capacity

           All the systems work for you, but at low usage level. This means that you are spending money on extra computing resources that will not be used under normal load and operation condition

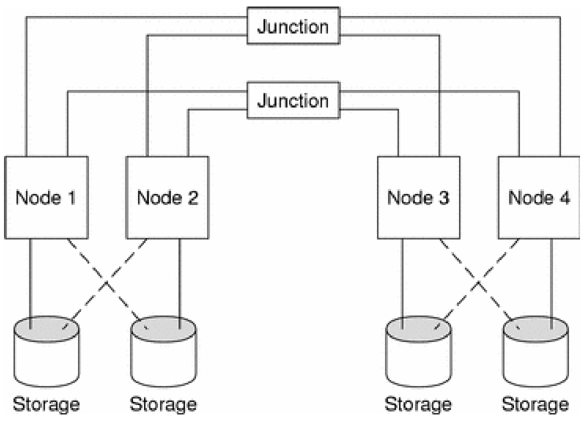
        ii. Maintaining a stand-by server

            You are spending money on a system that does no work for you unless it is needed as a failover server

4. CLUSTERS
    a. Also minimize the likelihood of a system FAILURE
    b. Clustering will help a system to be more scalable and High Available (HA)
    c. Cluster is a group of server systems and support software that is used to manage the server group
    d. Cluster provides high availability to system resources
    e. Cluster software allows group administration
    f. Cluster software detects hardware failure
    g. Cluster software detects software failure
    h. Cluster software handles systems failovers
    i. Cluster software automatically starts services in a event of failure
    j. There are different cluster configurations that are available
        i. Two Node Cluster (Symmetric and Asymmetric). You can either run both servers (Symmetric) or run only one server and keep the other as the hot standby (Asymmetric). This is the minimum high availability cluster that can be built

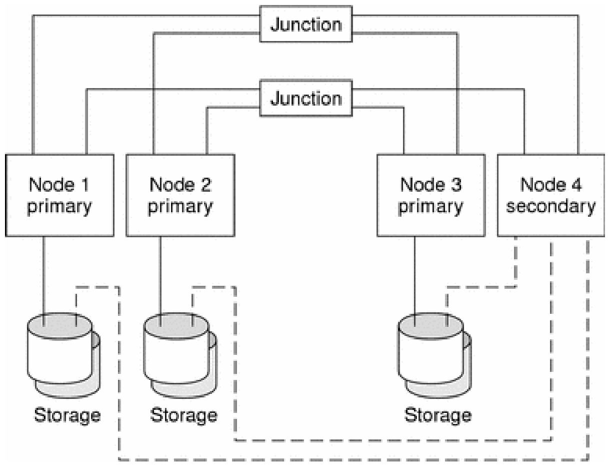


ii. Clustered Pair – places two machines in to the cluster, and then uses two of those clusters to manage independent services. Uses to manage highly coupled data services , such as an application server and its supporting data base server



iii. Ring – a configuration topology that allows any individual node to accept the failure of on of its neighboring nodes (not supported under SUN Cluster)

iv. N+1 (Star) cluster – N independent nodes and 1 backup node to which all clusters would fail over. The back up must be able to handle all the failed ones



v. Scalable (N-to-N) – A configuration that has several nodes in the cluster and all nodes have uniform access to data storage medium.
    1. The Data storage medium must support the scalable cluster by providing a sufficient number of simultaneous node connections

5. IMPROVING PERFORMANCE

a. Important to distinguish two types of times

    i. **Processing Time** – Times spent in computing , marshalling , un marshaling , buffering and transporting data
    ii. **Blocked Time** – Lack of resource , dependency of other processing

b. Ways to increase the PERFORMANCE of a system ,
    i. Increase the system capacity by adding more raw processing power
    ii. Increase the computational efficiency by employing efficient algorithms
    iii. Introduce cached copies of data to reduce the computation overhead
    iv. Introduce concurrency to the computations that can be executed in parallel
    v. Limit the number of concurrent request to control overall system utilization
    vi. Introduce intermediate response to improve the performance perceived by the user

vii. To improve the throughput it is advisable to have timeouts for operations which involve external systems

6. IMPROVING AVAILABILITY
   a. Factors that effect system availability are
      i. System downtime
      ii. Long response time
   b. Most common practice to improve the system availability is through replications
      i. Active Replication

         1. The request is sent to all the redundant components which operate in parallel, and the only one generated response is used. In active replication system downtime is short since all the component are actively synchronized

      ii. Passive Replication

         1. Only one of the replicated components (PRIMARY) responds to the request. The state of other components (SECONDARY) are synchronized with the PRIMARY component. In the event of a failure the service can only be resumed if a secondary component has a sufficiently fresh state

7. IMPROVING EXTENSIBILITY
   a. The need for extensibility typically originated from the change of a requirement
   b. Clearly define the scope in the service level agreement
   c. Anticipate expected changes. You should identify commonly changed areas of the system and isolate these areas into one coherent component. This will minimize the RIPPLE effect
   d. Design a high quality OBJECT MODEL –The object model of the system has an IMMEDIATE impact on its extensibility and flexibility. Apply essential object oriented concept , principles and appropriate architectural design patterns to your architecture

TIERS IN ARCHITECTURE

1. An architecture can have multiple LOGICAL tiers such as client , web service and data base server , but deployed in same physical server
2. With the advent of virtualization , the physical location has become immaterial
3. When talking about Two Tier , Three Tier , or N – Tier the client tier is usually not included unless explicitly mentioned as two tier CLIENT/SERVER
4. TWO TIER systems are traditionally CLIENT / SERVER systems
5. Most TWO TIER systems have THICK client that includes both PRESENTATION and BUSINESS LOGIC and a DATA BASE SERVER
6. The presentation and business logic were typically tightly coupled
7. You could also have a browser based TWO TIER system with business and database on the same server
8. TWO TIER > ADVANTAGES
    a. SECURITY – HIGH – Most of the systems are behind the cooperate firewall
    b. PERFORMANCE – HIGH – unless the company uses very old laptops that have minimum memory
9. TWO TIER – DISADVANTAGEOUS
    a. AVAILABILITY – BAD – if one component fails , then the entire system is unavailable
    b. SCALABILITY – BAD – The only component that can be scaled is the DB
    c. EXTENSIBILITY – BAD – In order to add new functionality you will effect all others , hence extensibility fails
    d. MANAGEABILITY – BAD – You can not monitor all the clients
    e. MAINTAINABILITY – BAD – Has the same problem as EXTENSIBILITY , to change one it effects all
10. TWO TIER – NO EFFECT

   a. RELIABILITY – Not an advantage or disadvantage. As the load increases most of the DB servers can be reliable

THREE TIER and N-TIER SYSTEMS

1. THREE TIER systems are comprised of WEB, BUSINESS LOGIC and RESOURCE TIER.
2. MULTI TIER systems are comprised of WEB , BUSINESS LOGIC , INTEGRATION and RESOURCE TIER
3. They share the same advantage and disadvantage when it comes to Non functional requirements
4. THREE TIER / N – TIER > SCALABILITY – GOOD –
   a. Scalability is improved over two tier system since now presentation logic and business logic is separated.
   b. The business logic is now running on a server which can be CLUSTERED
5. THREE TIER / N- TIER > AVAILABILITY – GOOD –
   a. Availability is improved over two tier system since now the tiers can be clustered and can be provided failover
6. THREE TIER / N – TIER > EXTENSIBILITY – GOOD –
   a. Extensibility is improved over two tier since now functionality is separated in to tiers. A change to once component will not have a ripple effect on the others
7. THREE TIER / N-TIER > MAINTAINABILITY – GOOD –
   a. Maintainability is improved since the functionality is separated in to tiers now. You could modify presentation with minimal or no effect to business logic
8. THREE TIER / N – TIER > MANAGEABILITY – GOOD –
   a. Greatly improved since the tiers are deployed on Servers and these servers can be monitored
9. THREE TIER / N- TIER > SECURITY - NON OF TWO –
   a. Allows for more points to secure
   b. But can effect performance due to more points
10. THREE TIER / N- TIER > PERFORMANCE – GOOD / BAD both
   a. Since different component can work on its own job the performance may go high. But at the same time

performance could go low if so much of data should be transferred from one tier to another

11. THREE TIER / N – TIER – DISADVANTAGE – INHERATLY COMPLICATED