Conclusion, Recommendations and future perspectives

My research work was based on the further initiatives and improvements in the field of virtualization to achieve the high availability in the field of information technology. Virtualization has been improved a lot in the last few decades but still have some existing issues in the same field. Physical virtual machine failover is the most important and critical point at this point because there is no proper existing way currently to make a better error free virtual machine failover. There are lots of possibilities to improve the term load balancing, failover and virtualization. We can make the application load balancing better further introducing the active and passive hardware load balancer layers between the client and the backend servers and failover can be improved in a lots of new manners.

Currently there is no proper way to make primary machine and secondary failover machine in proper synch for the better failover. There are lots of ways and algorithms developed for inter communication between primary and secondary machine but still this is a hot issue in the field of information technology. If we go in my approach then we can build up a mechanism of heartbeats which will provide the communication between the primary and secondary failover machine and in case of communication breaks by the primary machine then the migration process will start to the secondary machine. To be in synch, primary machine will send a heartbeat to the secondary machine to inform that I am available and secondary machine will acknowledge the signal in every 2 or 3 seconds and if secondary machine is not able to get three or defined consecutive signals from the primary machine then the migration from primary to secondary will start but still have some issue with this mechanism like sometimes if there is some network breakage between the primary and failover machine and they are not able to communicate with each other then however the primary machine is working then in that case the migration will also start even your main machine is working properly. So apart from the heartbeat mechanism we need to figure out some alternate and perfect solution for the 100% correct migration. Since the network is the only communication channel between machines and servers so we can develop a mechanism like the secondary failover machines can be shared between several primary machines and for the proper communication between main and failover machine, we can make the communication via different sub nets so that in case of problem on a particular sub net the another will able to communicate with the primary machine. But in that case also we have to make sure that only one sub net communication is active at one time and secondary will active only when the primary one will not able to contact backend machine.

So organizations are started accepting virtualization worldwide but still have some open issues are there and by developing many new systems and mechanisms we can make this field further better so that all organizations should start taking benefits of virtualization without any hesitation.
Better failure detection, Failover and Recovery

Till today there is no proper way to identify the failure of primary machine and with respect to that the failover capabilities to the standby machine is still a big challenge in the field of virtualization. Sometimes it happens that the primary machine wouldn’t crashed properly and the deployed application also did not respond and since the standby machine wouldn’t be able to get the failure acknowledgement so failover didn’t happened. By developing a hearbeat mechanism we can create a new solution to identify the health of primary machine within proper defined time frame, like suppose will develop and send a signal to the main primary components of the primary machine and will look for some predefined pattern to make that for that particular part health is good and at last with respect to all components heartbeats signal it will identify automatically if the primary machine is running and responding properly or not and if failover is really required or not. Component monitoring will based on the hardware components as well as the application or services deployed on the primary machines. Suppose if we have multiprocessor machine with around 16GB of RAM and we have certain applications running over there with some application url’s out of which applications using some tcp as well as udp port then, will design some mechanism which will monitor the components of hardwares, each url and as well as each tcp & udp port to gather complete status of applications as well as hardwares, after that it will match the collected results with the pre stored collections and if in case the collected metrics results are more than 60% is not according to the desired results than it would assume the failure of the primary machine and will start the migration failover process from primary machine to the standby server.

Apart from the above scenario, there would need to find the alternate solution of monitoring the failure in case of the failure or any other network congestion or latency. Since any existing or newly proposed solution is based on the certain heartbeat mechanism and all this rely on the network traffic completely and in case of any latency or failure on network this process will break down completely. So there would be another alternate solution need to identify to detect the failure in case of network failure. I would purposing a solution here like to use multiple NIC card on the primary and standby machine and each both primary and secondary machines should connect via different NIC cards as well as via different sub nets to provide the failover capabilities in case of failure or latency on any one of the network the monitoring will happen from the other subnet channel.

So the complete solution to detect the proper failure of primary machine would depend on many factors where first we need to collect the existing hardware and software information and default behavior, information about the used TCP and UDP ports, about the network setup and about all existing subnets. After that a solution need to define
which will collect the default behavior of each and every component within a proper defined interval and save the information to a particular place for further review in the future.

**Better cluster load balancing algorithm**

Till now we have three different major defined mechanisms for load balancing algorithms there in the field of IT software application server like in weblogic server but still we have some limitations and drawbacks there related with the existing algorithms and for which we need to identify a proper cluster load balancing algorithms.

Three existing cluster load balancing algorithms are Round robin, Waeitage based and Random but with respect to the benefits there is a major drawback associated related with each algorithm. Round robin routes the requests one by one to each member of cluster and once max limit reached it again started sending request from first one member to last one again in cyclic manner. Like suppose if you have 4 members in the cluster then any request that will come to cluster will first goes to first member, second request to second member, third request to third member and fourth request to fourth member, and in case of further next request, it will divert to again to first, second, third and fourth member in cyclic manner. In that case it maintain the loads equally across each cluster member but the drawback is, if suppose we have four servers in the cluster and in case any one or more servers are under heavy load and not able to handle extra requests cluster will still try to send request the respective server and due to overloaded the that particular server or servers will be crashed. Basically this load balancing is managed by the plugin modules provided with each and every application servers and web servers are configured to forward request to the respective clustered members. I would recommend to define or update the logic inside plugin modules of application servers to first identify the load of each and every server with respect to the each cluster member and the forward the incoming request to the cluster member which is under less load. Like suppose we have four cluster members and for each cluster member we have defined 1 GB or RAM which is required for JVM to run application objects. During processing of requests suppose a new request came to web server and before forwarding request to cluster, pluging will first identify the free JVM of each and every server and forward the request to the server having less used JVM according to other three members of the cluster.

In Random algorithm, requests from web server diverted randomly to any of the existing cluster member. The drawback is same in case of any of the server is under heavy load the cluster will still try to send the request to the same and even in that might be the less loaded server will get very less request and the server under heavy load will still get the lots of requests and due to heavy load the server will crash so here I would recommend to define or update the logic inside plugin modules of application servers to first identify the load of each and every server with respect to the each cluster member and the forward the incoming request to the cluster member which is under less load.
In Waeitage base the requests diverted from web server according to the defined cluster member Waeitage, like suppose we have two cluster members and one cluster member is on host A with less hardware configuration with respect to the other cluster member which is on host B which is with high hardware configuration so the more requests will diver to second cluster member in respect to the first one. But the drawback is still same in case of any of the server is under heavy load the cluster will still try to send the request to the same and even in that might be the less loaded server will get very less request and the server under heavy load will still get the lots of requests and due to heavy load the server will crash so here I would recommend to define or update the logic inside plug-in modules of application servers to first identify the load of each and every server with respect to the each cluster member and the forward the incoming request to the cluster member which is under less load.

**Better session management & Replication**

We have the existing session management solution exist in the market but still have some drawbacks are there or you can say shortcomings are there. Session replications are the sessions or the information any client can create during the access of any application and a better management of sessions are required to maintain the session information of a client when he would connected with a particular member of the cluster and in case of that member got failed or crashed due to any reasons the session information of that particular client should replicate to any one of the existing member of the cluster.

Existing session replication solution still have the shortcomings like all the sessions of a particular client created on the same JVM of the cluster member on which the client session connected. Like suppose we have two members in the cluster and any one of the client connected with the first member then any of the client sessions would create on the same client member JVM and it creates the issues because sessions take large amount of the JVM size and lots of time it cause the server to crash due to shortage to the JVM for any particular cluster member. So in this way if you are using the large web applications which required large client session caused frequently server crashed or hang problems due to large sessions on the same JVM which is in used by the server for application objects. I have purposed a alternate solution for this here like we can define a separate JVM apart from the cluster server members to handle the sessions of the client in case of you are using a large web applications since small session applications can be handled by the JVM but purposed solution still beneficial in this case also. In case of the existing session replication solution we have to modified the code of application also in case to make application replicate but in the purposed solution we can define a separate JVM and for which we don’t need to define and modified code every time, Will define a unique application which will be deployed on the application server and a separate JVM with good heap size will be created and each and every session will maintain in the same pool and this will also avoid the replication group kind of things which need to defines at the application level.

In the existing solution there is one and only one group can be defined for the session replication and in case of any issues with the secondary replicate group the complete client session will be lost. Like suppose if we have four cluster members and we have defined replication group RG1 associated with second cluster member for first cluster member then all the sessions of primary cluster member will replicate on the second
cluster member but in case second cluster member will down due to any reason simultaneously with the first one then complete session of the client will also loss. In the proposed solution all the session will maintain in the separate sessions and all the cluster members will share the sessions of each other so that in case of problem with any one of the cluster member any one of the available cluster member will serve request and will handle the client existing session.

So we can used the purpose solution in case any one of the below condition

- If our application handling very large sessions of HTTP
  
The purposed solution will handle the large http sessions outside of the cluster member JVM and will avoid any kind of memory crunch or exhaust problem.

- We are getting memory ( JVM ) constraint issues due to the large sessions
  
The proposed solution will avoid the memory issues since session will be handle by the external jvm.

- If we want a separate JVM for http sessions so that each and every session will replicate for each and every cluster member.
  
The purposed solution will avoid the any unnecessary modification on your applications to enable session replication, a standard process will be developed to make it ready to use anywhere without any modification on the existing code.

- Cluster members can be start without fear of losing the session data
  
Since all the sessions will be handled by the external JVM so each and every cluster member can be stop and start any time without fear of losing the session data.

**SPOF (Single Point of Failure) With Virtual Shared I.P Address**

As of now we are using the static IP address or you can say static DNS as the listen address for each and every web server in the IT infrastructure which is the single point of failure. Sometime we used the web servers in the cluster way also but since there are limited number of web servers in comparison with the application servers so in case of any cluster web server crashed or goes down due to any reason then complete load will divert to the only available cluster members and due to heavy load it may cause the existing server may also goes down.

To avoid single point of failure issues with the web server in IT infrastructure I have purpose a alternate solution for the listening address to use the virtual shared I.P address for the web server in case of using the static address, This will provide the
capabilities to the web server to migrate from one physical server to another physical
server in case it goes down on the primary server. Using shared virtual I.P technology, a
virtual shared I.P address shared across multiple physical machines and in case of any
issues with the primary machine it will migrate to any of the other secondary define
machine and will serve the request without any issues and without making other cluster
member overloaded. Of course apart from using the shared virtual I.P address instead of
Listening address we have to define some mechanism for the migration as we have
other existing web server modules we can define a separate module to identify and
migration of the web server process.