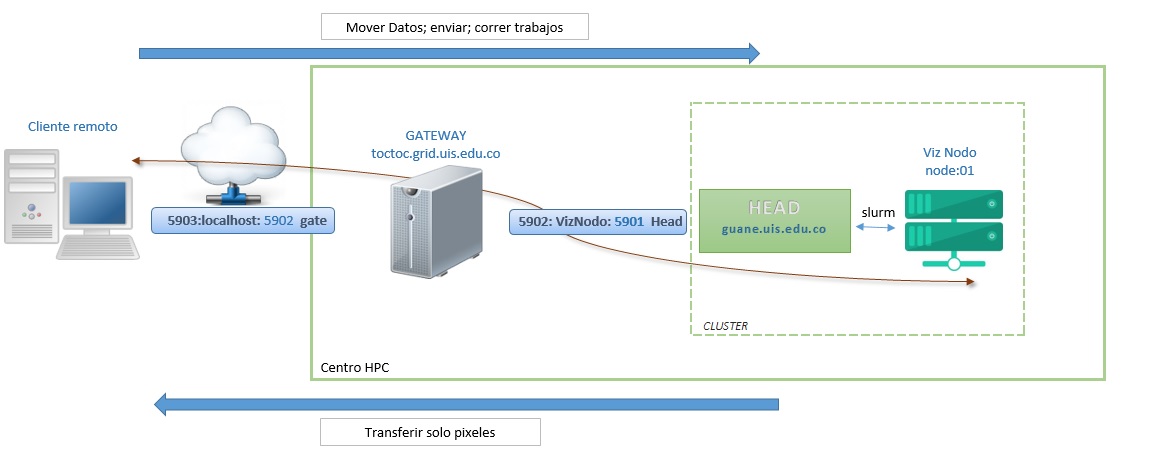
1. Esquema general de acceso al nodo de visualización mediante túneles ssh ( la hice como introducción y ventajas sobre la arquitectura presentada, mostrando a grandes rasgos cómo es la arquitectura actual )



la metodología se basa en el uso de contenedores como servicio CaaS, un paradigma reciente que se suma a servicios como: Aplicaciones como servicios AaaS, plataforma como servicios PaaS, e infraestructura como servicios IaaS.

Una ventaja de CaaS sobre Paas es que los usuarios pueden obtener la misma experiencia sin gasto significante de dinero utilizando su propio hardware o cuentas en la nube, empaquetado en contenedores todo lo necesario para desplegar sus proyectos y así no estamos limitados a un proveedor en particular como lo es Amazon con sus web services.

\*----- NOS DA LA OPORTUNIDAD DE TENER ENTORNOS DE DESARROLLO DE PRUEBA CERCANOS A PRODUCCIÓN----\* corrigiendo así algunos fallos al implantar software en entornos heterogéneos, todo ello para disminuir el uso de una frase clásica .... en mi equipo funciona. ...

Hay tres sabores en el área de virtualización a saber:

-máquinas virtuales (virtualización de maquinas, todo tipo de hypervisors)

-paravirtualización (modificación de binarios para hacerlos compatibles)

-contenedores (virtualización de entornos)

\*\*\*busco hablar un poco de cada una de ellas, y especificar el porqué seleccionamos contenedores. basándose en características, diseño, ecosistema, portabilidad, complejidad de despliegue y administración\*\*\*

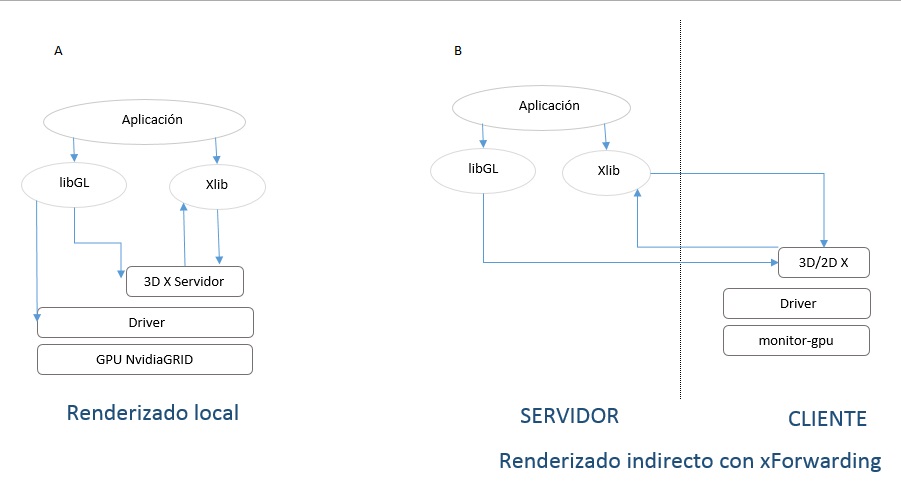
2. por medio de los siguientes esquemas, pretendo explicar la ventaja de uso de una librería interpuesta como lo es Virtual GL y un proxy Xserver (turbovnc), sobre el estándar de visualización clásico local y la visualización de xForwarding ( ssh -XY guane.uis.edu.co) que no es eficiente para el caso. *(los diagramas son propios, basado en lecturas* ). Para ejemplarizar conceptos ilustró componentes individuales de una aplicación openGL.

El sistema de renderizado de una aplicación en un sistema \*nix se basa en un modelo cliente servidor usando varias formas de comunicación de procesos entre la aplicación y el hardware.

\*se configura un sistema Xwindowing de la siguiente forma:  
xserver - encargado de manejar los recursos de hardware y proveer las interfaces a la aplicación

xlib librería - encargada de servir de pasarela entre la aplicación y el X server. (encapsula detalles del protocolo X)

Si una aplicación necesita usar openGL, primero pregunta al servidor X para crear un contexto openGL, una vez el contexto está listo, openGL puede hablar directamente a la GPU, saltándose la comunicación con el X server. **RENDERIZACIÓN LOCAL imagen A**

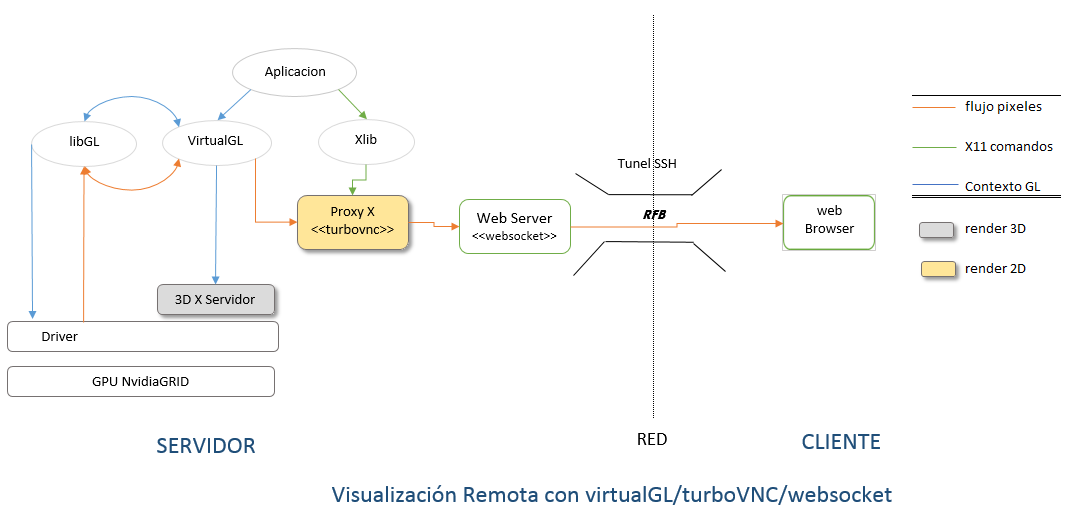


Esta arquitectura permite extenderse a una configuración remota **RENDERIZADO INDIRECTO CON xFORWARDING imagen B ,**  requiere un X server en el lado del usuario “cliente” , la aplicación intenta usar las características openGL (contexto ) disponibles en el servidor X, que corre en una máquina remota; de esta forma se usa la GPU del lado del cliente.

esta metodología penaliza de un modo significativo el rendimiento de la aplicación, aún más si es interactiva, puesto que requiere comunicar el contexto openGL, comandos X11 para el control de ventanas y la interacción con el usuario.

utilizando técnicas de compresión sobre el protocolo X, se puede mitigar en algunos casos este impacto ( noMachine), pero no es suficiente para aplicaciones que requieren alta demanda de visualizaciones, pero esto no permite tomar ventaja de la GPU remota, tornándose en un caso de renderización por software.

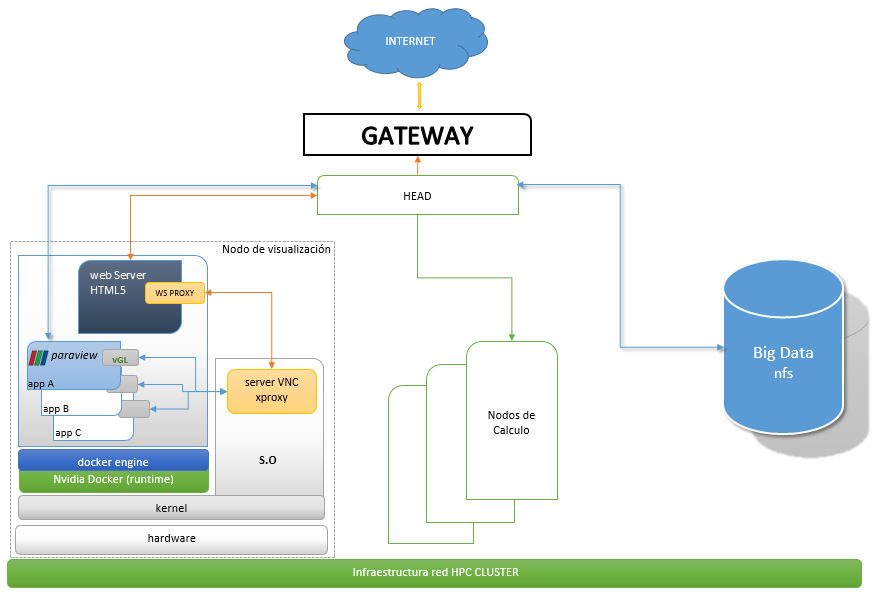
Se propone implementar la siguiente metodología:

En este punto se ha detectado un cuello de botella en la arquitectura como lo es la transferencia a través de la red para el proceso de renderizado; para afrontar este aspecto se busca que la comunicación se realice en su mayoría del lado del servidor incluyendo el contexto openGL, manejo de ventanas, y la interacción del usuario; dado lo anterior se puede implementar una solución de alto rendimiento para el transporte de imagenes finales al cliente como son los modelos de compresión. 

Una tecnología clave es el uso de así llamado librerías interpuestas virtualGl, todas las operaciones son enrutadas a través la librería virtualGL, esta inspecciona las llamadas a openGL y actúa sobre las mismas o delega a libGL esta tarea cuando sea necesario. por ende virtualgl conoce cuando la gpu necesita escribir pixeles en un buffer de video, captura esta información y la reenvía a un servidor proxy que tiene por objeto manejar la compresión del contenido final y transmitir el contenido del buffer al usuario en una estación remota.

*Lo anterior permite que un usuario en una terminal con recursos limitados, tenga acceso a aplicaciones que requieren un entorno de alto rendimiento para ser ejecutadas.*

3. esquema general de la adaptación del nodo de visualización sobre la arquitectura ya presente, especificando que hay dentro del contenedor y cómo se comunica con los demás recursos del cluster.



-la comunicación entre los nodos de cálculo y el nodo de visualización solo se realiza mediante el servidor nfs, indicando así que si el usuario desea una sesion para visualización sobre sus datos primero debe realizar la simulación, almacenar sus datos y solicitar recursos de renderizado a través de una sesión interactiva de slurm.

4. Esta imagen representa el proceso general de una simulación en OpenFOAM, el objetivo de este trabajo es centrarme en el proceso de visualización remota, y el renderizado in situ cluster, aprovechando así el poder de procesamiento.

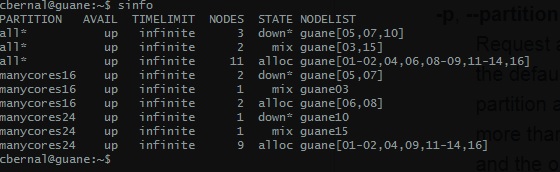


1. luego de que el investigador realiza la formulación del problema, el mallado y establecer condiciones iniciales procede a resolver el modelo físico por medio de ecuaciones numéricas con métodos predefinidos. Llama al solver icoFoam (openfoam) en esta fase ( generando así archivos base que servirán de DataSet para poner en marcha este proyecto en la fase de post procesamiento)

*$icoFoam.*

1. En la etapa de post procesamiento solicita recursos de renderizado. (una idea general era agregar el nodo de visualización a una partición marcada como interactiva de slurm y bloquear los demás nodos para que no se pueda llamar en ellos una sesión interactiva, como funcionalidad experimental)

*$srun --partition=interactive --mem=8000 --gres=gpu:ngrid:1 --pty --export=ALL --ntasks=24 --time=02:00:00 /bin/bash*

**

-el usuario ejecuta el comando para sesión interactiva

-slurm garantiza el acceso al nodo a través de ssh

-el usuario llama al programa que requiere recursos gráficos en la misma terminal dispuesta por slurm.

-un script epílogo se asegura de limpiar y liberar recursos para su próximo uso luego de terminar el trabajo del usuario.

*(inicialmente no uso el comando salloc* *para localizar recursos puesto que a través de srun* *en un solo comando se puede localizar dichos recursos y lanzar un job evitando así errores de los usuarios o comandos incompletos. tampoco se usa script para que el trabajo no pase a cola y solo se ejecute si hay recursos disponibles en tiempo real)*

* *a futuro podría agregarse la opción --cluster si en lugar de nodo de visualización se usa cluster de visualización.*
* *considere usar particiones y no especificar número de nodos puesto que desde la configuración de slurm se podría limitar características como tiempo de vida, máximo de peticiones , numero de nucleos por nodo, y crear esa partición sin que se traslapen nodos y/o demás recursos, limitando así vectores de errores de usuario.*

por ejemplo en slurm.conf

*GresTypes=gpu  
NodeName=guane17 Gres=gpu:ngrid:2*

*# PARTITIONS  
PartitionName=interactive Nodes=guane17 Default=YES MaxTime=06:00:00 State=UP Shared=NO MinNodes=1 TotalCPUs=12 TotalNodes=1 MaxMemPerNode=UNLIMITED AllowGroups=GridUIS*