UNIVERSITAT ROVIRA i VIRGILI

DOCUMENTACIÓN SD2

Sistemas Distribuidos Task 2

Pablo Sierra Fernández
Jorge Pérez Álvarez
2021/2022
Doble grado BIOGEI

Índice

EXPLICACIÓN DECISIONES TOMADAS:

Resumen tareas planteadas:

Para iniciar esta segunda práctica, partimos de una versión funcional que implementa XMLRPC procedente de la práctica anterior, que contenía las clases *xmlrpc_client.py, xmlpcr_cluster.py y xmlrpc_server.py*

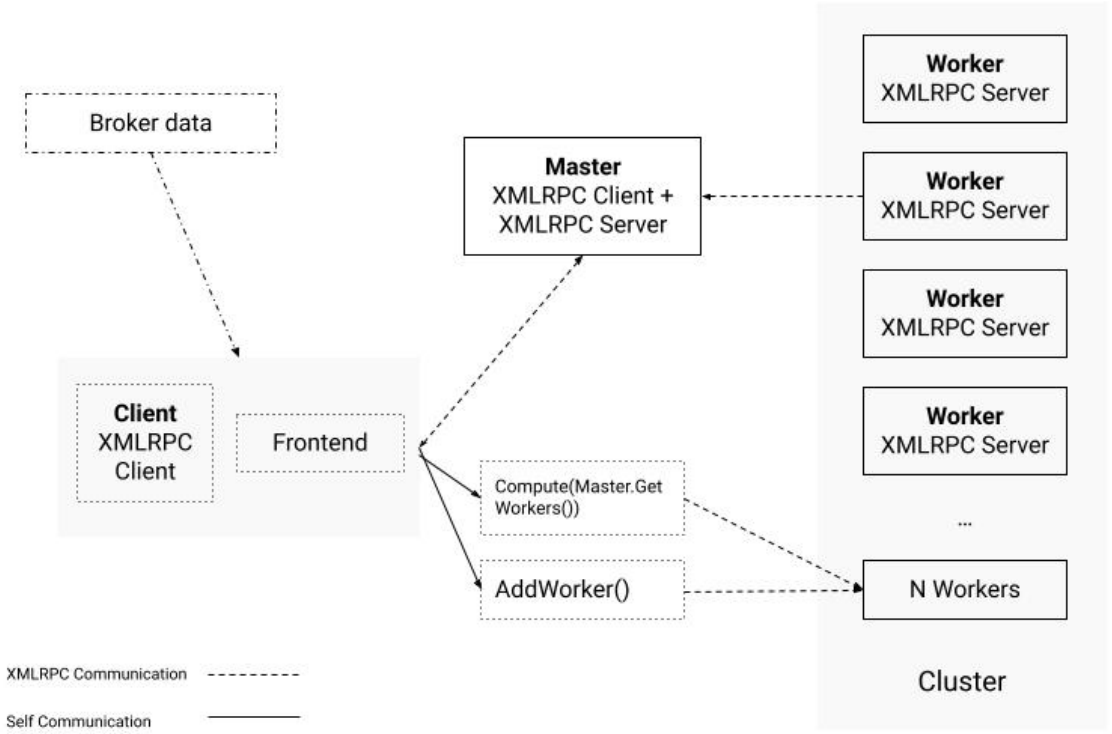


diagrama 1: XMLRPC del que partimos realizado en la práctica anterior

Para poder llevar a cabo las tareas de la siguiente práctica, decidimos implementar Redis. Los principales objetivos donde nos enfocamos se tratan de:

- failure-tolerance:

Durante la ejecución del programa tenemos diferentes workers y un cluster. El programa tiene que ser capaz de, si alguno de estos cae, esto no produzca fallos en la ejecución del programa. Para ello, nos centraremos en que todos los servidores conectados sean conscientes de que un servidor se ha caído y, en caso de que el servidor caído sea el cluster, otro servidor tiene que ocupar su función.
Como todos los servidores tienen que ser capaces de realizar las tareas de cluster y de worker, se ha optado por realizar una única clase, *xmlrpc_server.py,* que implementa todas las tareas necesarias de un worker y del cluster. A medida que se van iniciando servidores, el primero en iniciar siempre será cluster, si el cluster cae y hay más servidores dado de alta, será el primero de la lista el que se encargará de actuar como worker. En este punto fué donde se decidió implementar Redis para que todos los nodos sepan la dirección correspondiente al cluster.

- consistencia:

Para llevar a cabo la consistencia nos interesa que cada vez que se produzca un cambio (crea o elimina un nodo), tanto el cliente como los otros servidores tienen que ser conscientes del cambio que se ha producido y actualizar sus valores acorde con el cambio producido.
Para mantener la consistencia de los cambios en los servidores se decidió que cada vez que se eliminase o se crease un nuevo nodo, se actualizase la lista de nodos de todos los servidores en ejecución, esta práctica, además de brindar de consistencia, permite facilitar la tarea del failure-tolerance.
En cambio, para mantenerla en el cliente se implementó el modelo publish/subscribe por medio de Redis, donde los nodos envían un mensaje cada vez que se cree o elimine un nodo. El cliente, antes de realizar cualquier tarea, consulta los mensajes, si tenemos mensajes disponibles significa que hay que actualizar los valores en el cliente.

Diagramas:

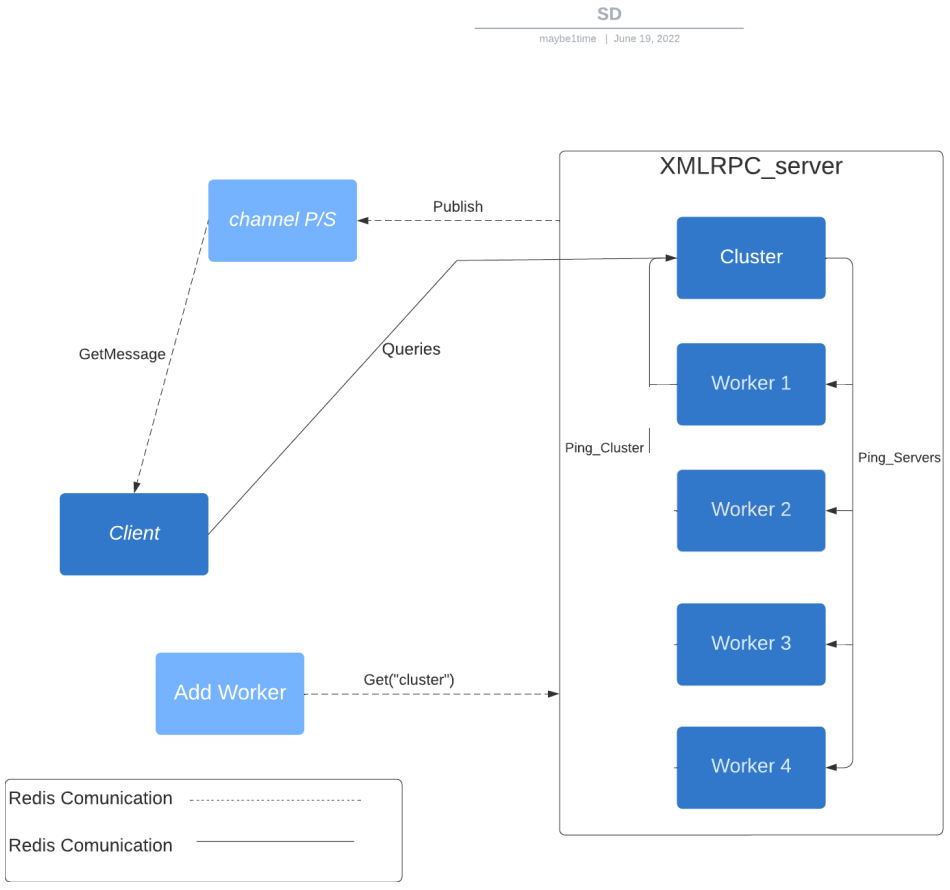


diagrama 2: failure-tolerance y consistencia

En el diagrama se muestra, en correcto funcionamiento como estaría estructurado el sistema.
En este, tanto Cluster como Workers son nodos de la Clase *xmlrpc_server.py,* donde como nuevas tareas del cluster se encuentra la función *ping_servers(),* el cual se encargará de

detectar cuando un nodo haya caído, en ese caso, se encargará de actualizar la lista de nodos conectados de todos los workers. Los workers en cambio tienen la nueva función *ping_cluster()*, donde el primer worker de la lista (Worker1 en el diagrama) solicita señal al cluster, desde que este no devuelva señal, Worker1 se encargará de cumplir la función de cluster y actualizar la lista de workers disponible de todos los workers.
En cuanto al empleo de Redis, se registrará cada vez que se modifique el cluster su dirección, de esa forma, cada vez que un nuevo nodo desee conectarse al sistema, consultará en Redis quién es el cluster. También implementa una comunicación Publish/Subscribe entre cliente y los nodos para informar al primero cada vez que se ha producido un cambio. Antes de realizar cualquier consulta, el cliente consultará los mensajes para saber si tiene que actualizar sus valores.

JUEGO DE PRUEBAS:

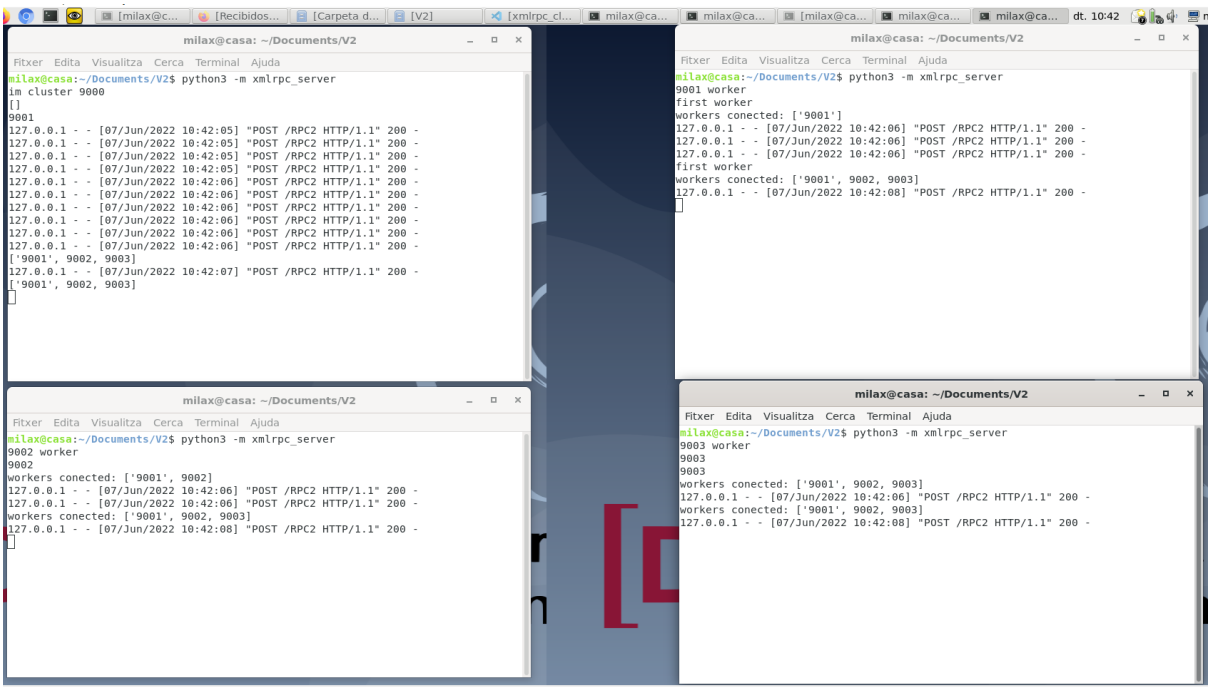
Para realizar el juego de pruebas, ejecutaremos la aplicación y la forzaremos a los casos más complejos en relación a los puntos tratados en la práctica.
Realizaremos las pruebas para testear failure-tolerance y consistencia.

Failure-tolerance:

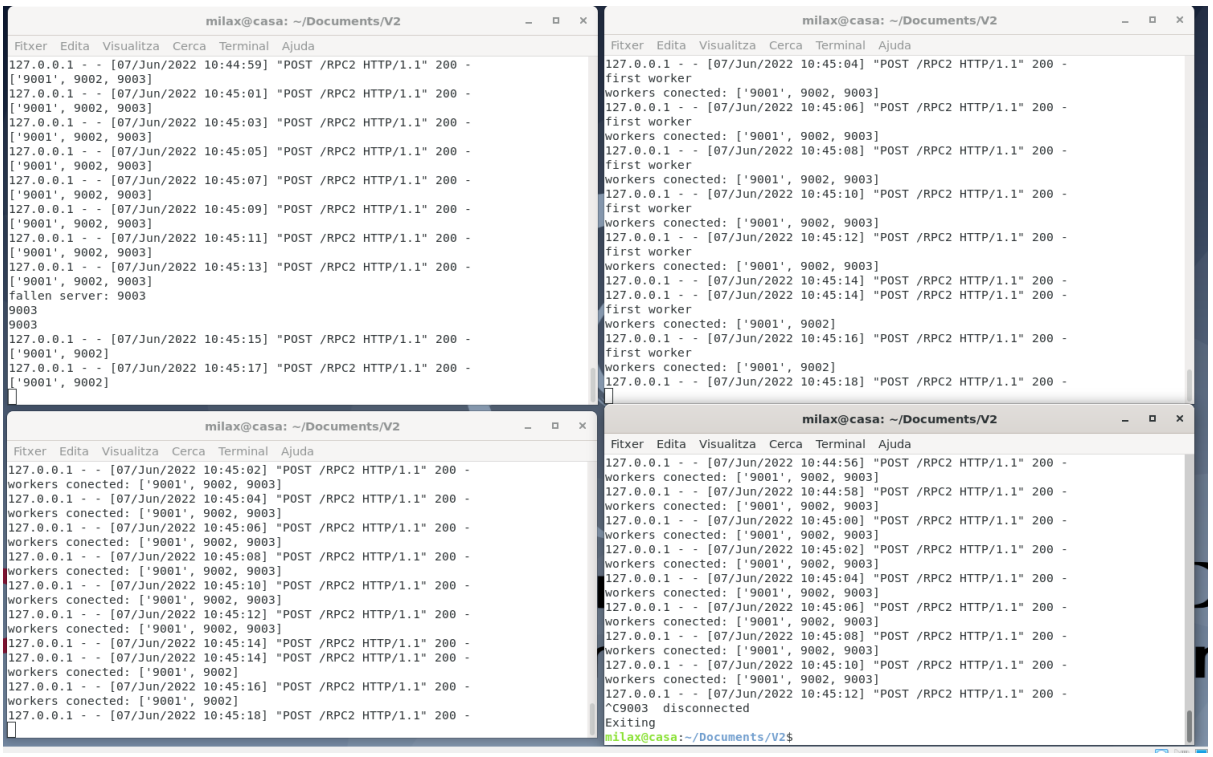
Para comprobar la respuesta correcta ante los fallos una serie de pruebas sin necesidad de ejecutar el *xmlrpc_client.py:*

- **caída del worker final:**

resultado esperado: todos los nodos actualizarán la lista de workers disponibles
en la primera captura vemos el correcto funcionamiento al ejecutar en 4 terminales *xmlrpc_server.py:* en la primera línea podemos ver como en la esquina superior izquierda tenemos el cluster, a su derecha el 1ºworker y el resto son otro workers

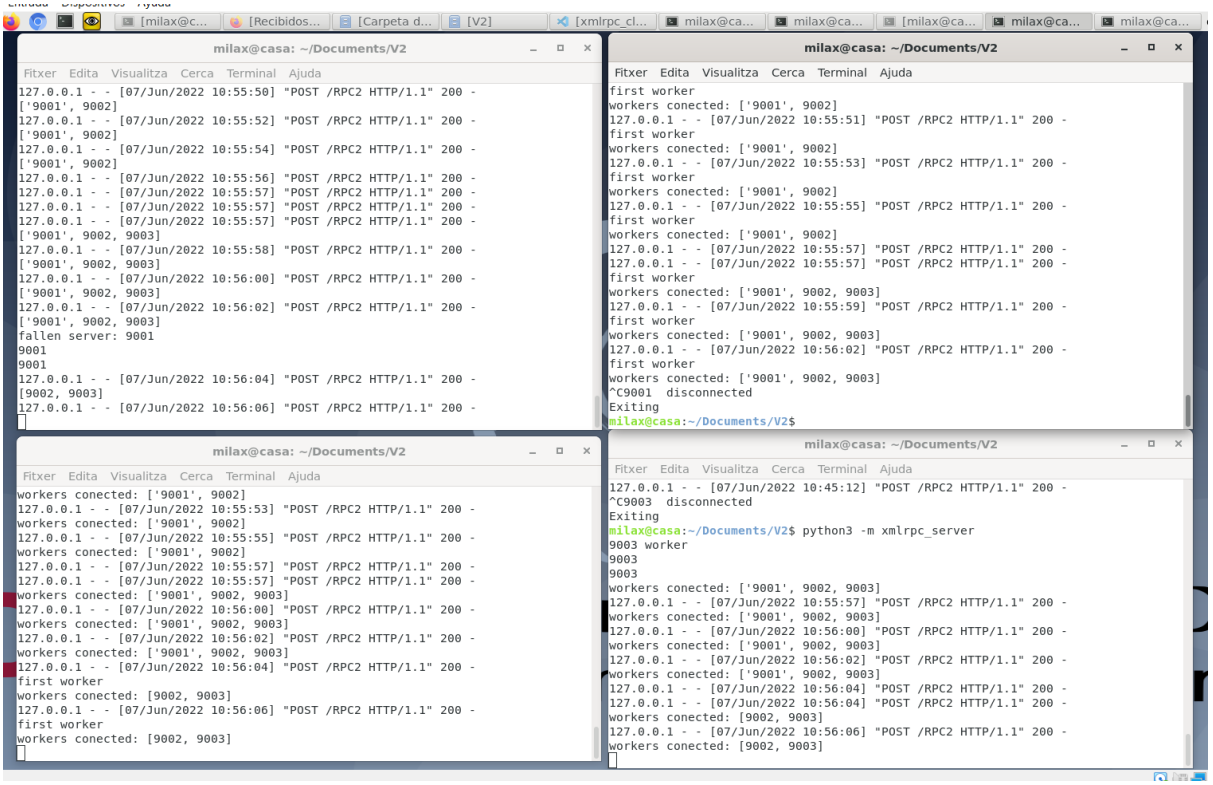


En esta segunda imagen terminamos la ejecución del worker 9003 y vemos como el resto actualiza la lista de workers eliminando al 9003.



- **caída del primer worker:**

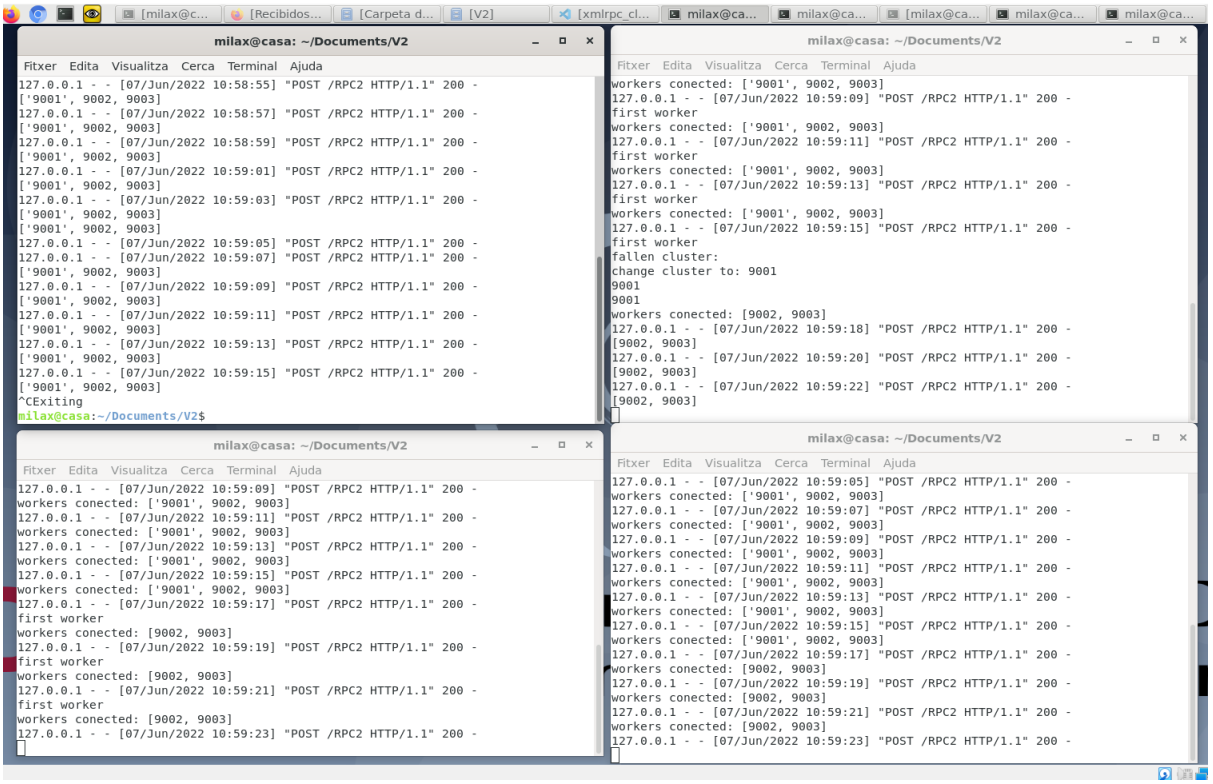
resultado esperado: todos los nodos actualizarán la lista de workers disponibles y el siguiente de la lista pasaría a ser el primero.



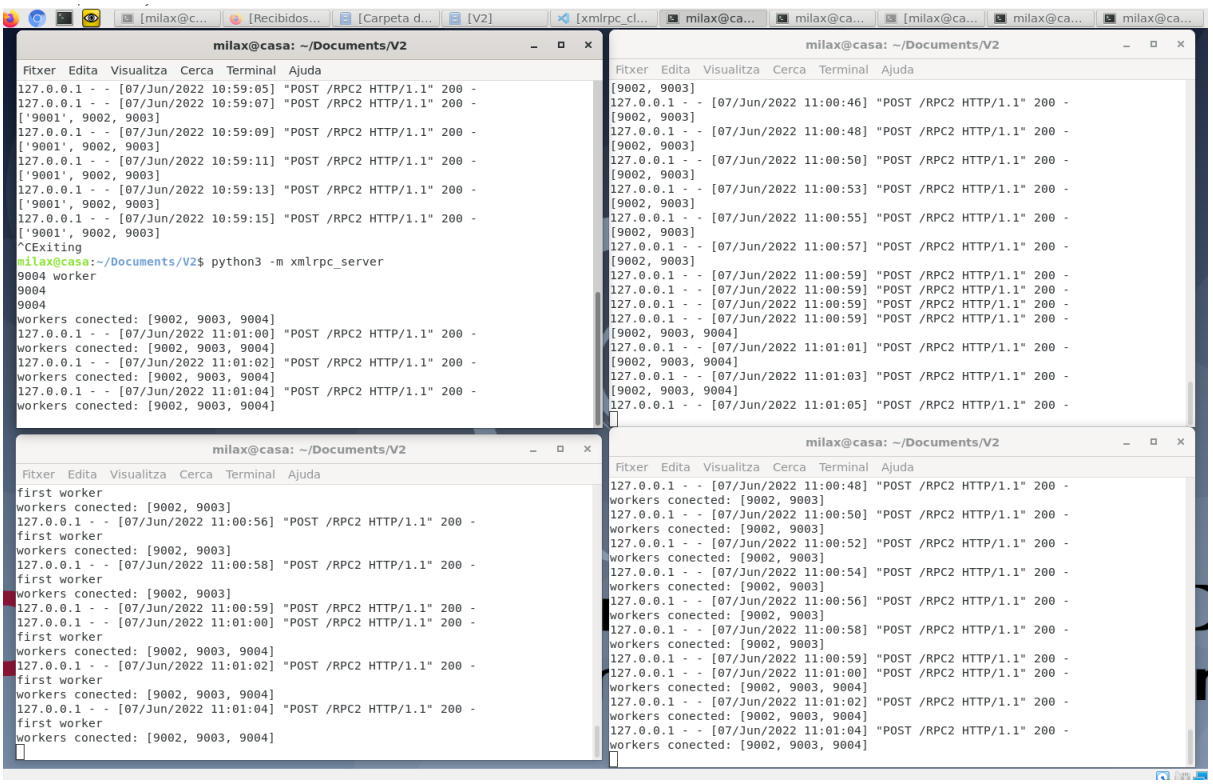
Como se observa en la imagen, se ha desconectado el worker 9001, se actualizan las listas y 9002 pasa a ser el primer worker de la lista.

- **caída del cluster:**

resultado esperado: El primer worker de la lista asumirá la función de clúster.



Como se ve en la ventana arriba/derecha correspondiente al worker 9001, desde el momento que ha terminado de ejecutarse el cluster, 9001 ha pasado a cumplir la función de clúster.



Si volvemos a iniciar otro worker vemos que sigue funcionando todo con normalidad siendo 9001 el cluster.

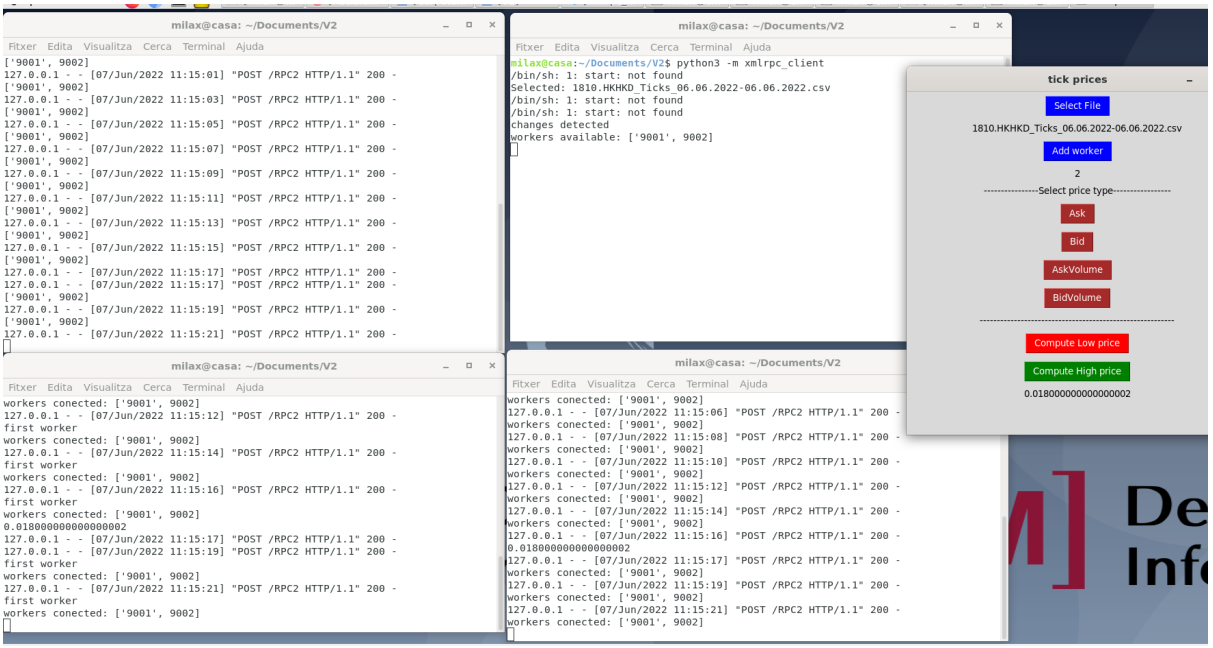
Por tanto, se podría decir que cumple correctamente la tolerancia a errores por caída de un nodo.

Consistencia:

Ya hemos comprobado la consistencia entre los nodos revisando las listas en la prueba anterior, faltaría comprobar la consistencia del cliente. Para ello ahora ejecutaremos tanto *xmlrpc_client.py* en 1 ventana y *xmlrpc_server.py* en 3:

- **Primera ejecución:**

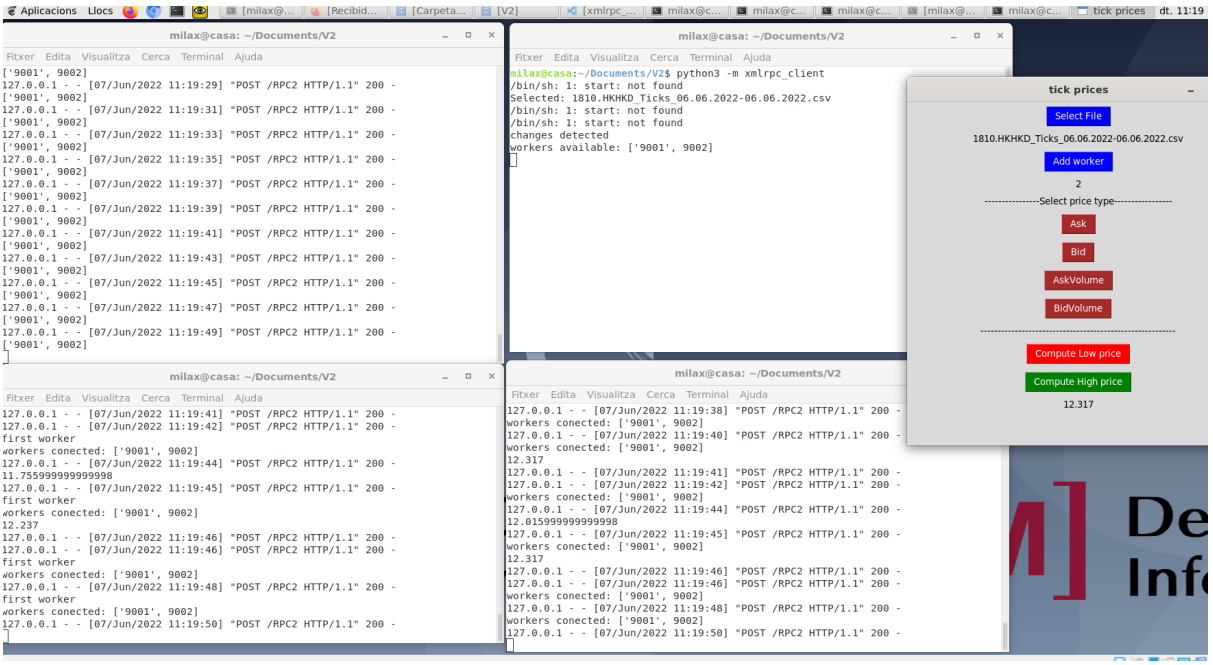
resultado esperado: el cliente calculará y repartirá el trabajo correspondiente a cada worker
Para comprobar cuando se actualizan los valores de cliente, mostrará un mensaje por pantalla.



tras en la ventana ‘tick prices’ seleccionar el fichero, clickeamos en ‘compute low price’ y vemos como por la terminal de la esquina superior izquierda, imprime “change detected”, por tanto a procedido ha hacer la correcta partición.
ahora si seleccionamos la otra operación simplemente se actualizará el valor al máximo y no mostrará nada la terminal.

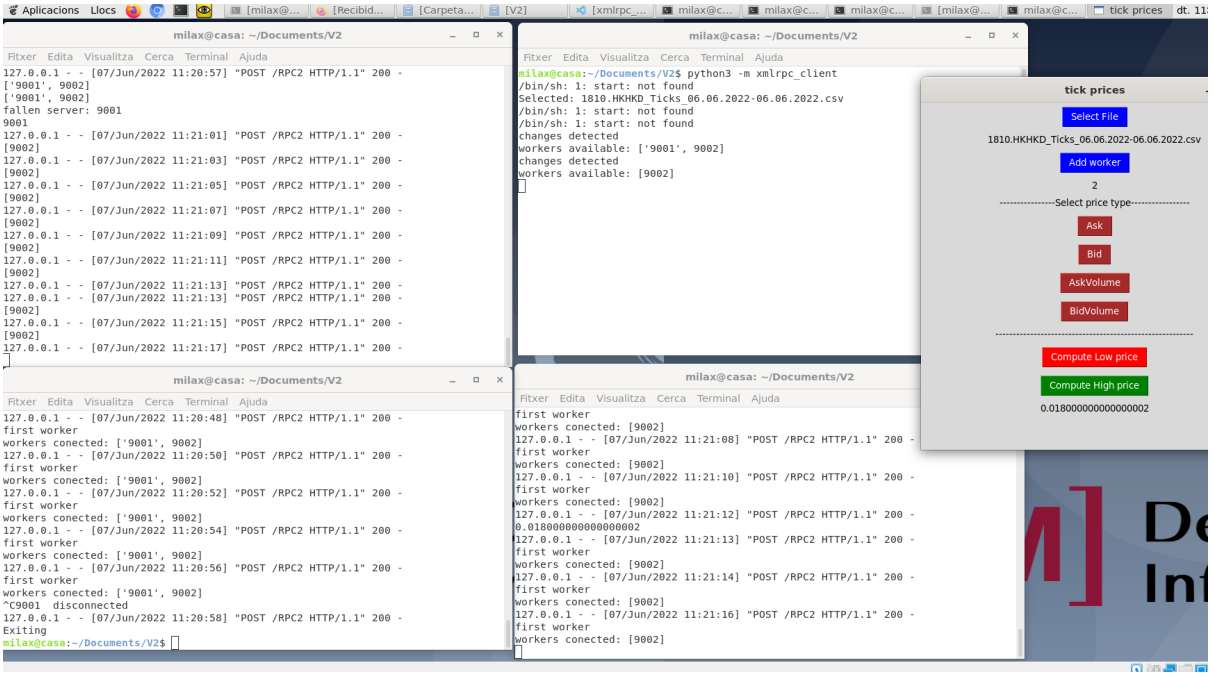
Conclusión:

Los problemas planteados han sido resueltos con éxito. No se ha realizado la tercera parte de Cloud implementation.



- **Caída de un worker:**

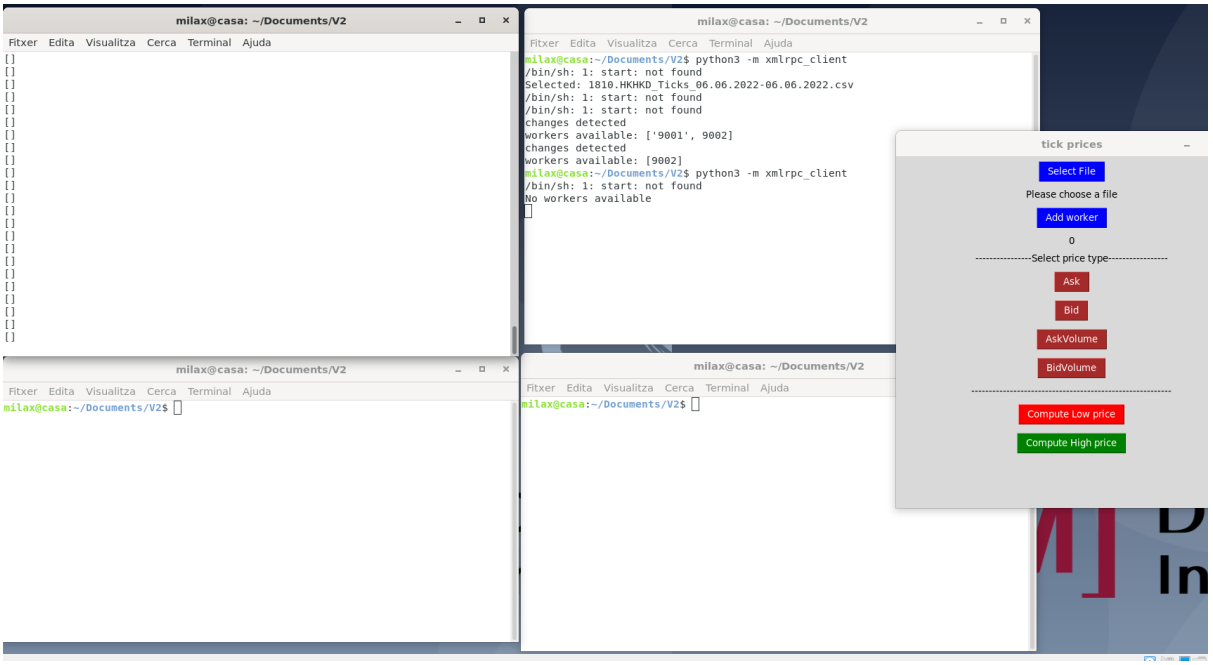
resultado esperado: el cliente calculará y repartirá el trabajo nuevamente con el nuevo número de workers disponible



Vemos en la terminal como ahora solo hay un worker disponible para la tarea y vuelve a repartir el trabajo

- **Casos de error:**
  - no hay workers disponibles

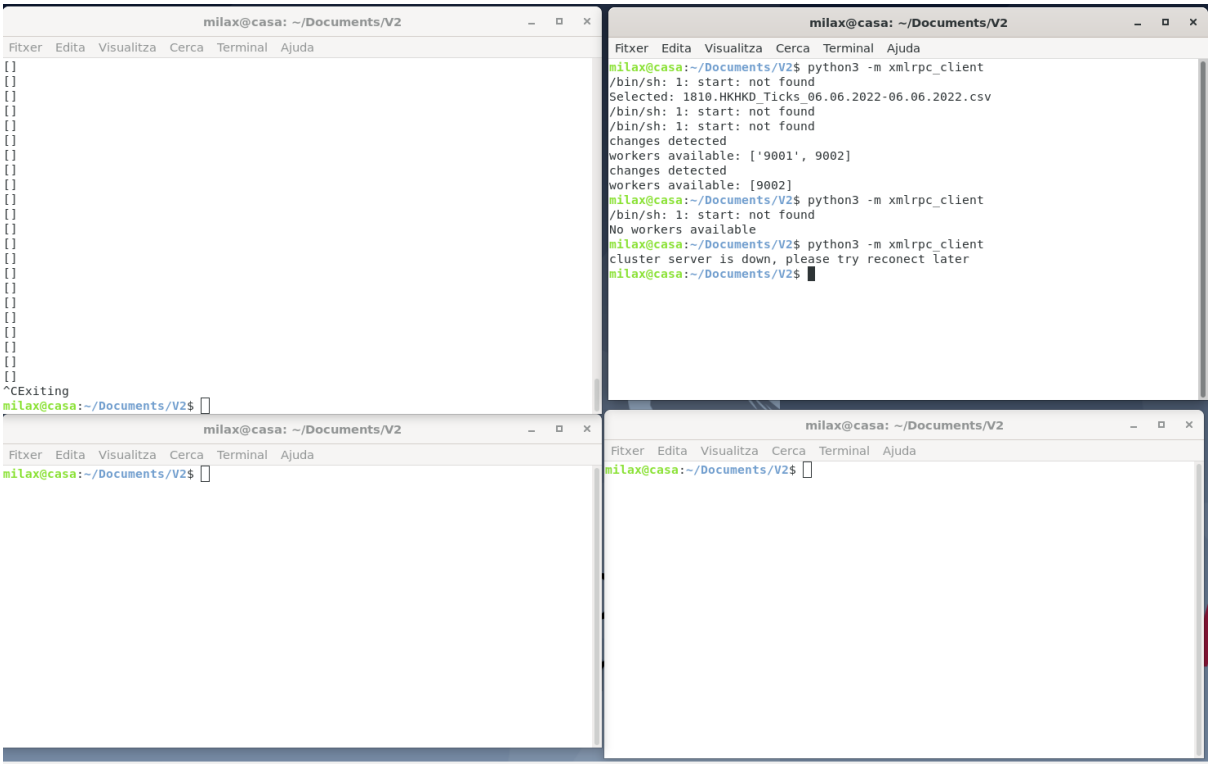
resultado esperado: mensaje por el terminal de que no puede hacer la operación



Resultado correcto

- no hay cluster iniciado.

<u>resultado esperado</u>: notificación y cierra el cliente



resultado esperado.

Para terminar hemos decidido realizar muchas operaciones: eliminar 2 workers seguidos, eliminar 2 cluster seguidos, iniciar más de 5 workers… Y no se observó ningún error inesperado.