**Informe experimento 2 Solumovil**

**Pre experimentación**

**Problemática:**

La ubicación y el seguimiento de un gran número de vehículos es un aspecto de vital importancia para la empresa de transporte Tbc. Por esto Tbc ha decidido contratar a SoluMovil para diseñar y construir una aplicación que sea capaz de responder ante un gran número de solicitudes simultáneas de sus vehículos (Mobibus, vcub y tranvía eléctrico) así como una respuesta rápida ante estas. Adicionalmente la aplicación construida debe atender todas las solicitudes que se le hagan.

**Objetivo del experimento:**

Aceptar, procesar y responder a las solicitudes de múltiples usuarios simultáneos en la aplicación SoluMovil con el fin de satisfacer las necesidades de la empresa Tbc. Específicamente, responder al 100% de las solicitudes enviadas por los vehículos manteniendo los atributos de desempeño y escalabilidad.

Descripción del experimento:

Para cumplir con los requerimientos del cliente se siguieron los siguientes pasos:

1. Terminación, corrección y pruebas de los requerimientos funcionales de la entrega del experimento 1. Esto fue completar los servicios REST así como implementar estos en las aplicaciones móviles.
2. Investigación de métodos de balanceo de carga así como de tecnologías que permiten su implementación de forma sencilla con Django y Python. Selección de una tecnología.
3. Implementación de la tecnología escogida en el paso anterior.
4. Pruebas de carga teniendo como métrica la disponibilidad. Está siendo del 100%.
5. Al no obtener los resultados deseados se repitieron los pasos 3 y 4 hasta que se cumplieron los objetivos.

**Artefactos a construir.**

Módulo de balanceo de carga en la aplicación. Aplicaciones móviles completamente terminadas. Servicio REST.

**Recursos de Experimentación**

Seleccionar los recursos no fue una tarea sencilla. Esto ya que se debía respetar los elementos de la arquitectura que proporcionaban la escalabilidad y el desempeño y a la vez implementar nuevos que satisficieran la disponibilidad.

**Hardware.**

Se utilizó un PC con sistema operativo Windows 10 y RAM de 4GB como servidor de la aplciiacón. Este fue el encargado de redirigir las peticiones a través de nginx.

Se utilizaron como nodos del servidor dos MAC con RAM de 4GB utilizando gunicor como servidor interno para desplegar Django.

**Resultados esperados.**

Se espera que cualquier petición al servicio REST sea resuelta en menos de 1 s además de que el 100% de las peticiones tengan respuesta y que 4500 peticiones tengan un tiempo de respuesta medio menor a 1s.

**Duración y etapas:**

AL hacer el planeamiento se tienen 3 etapas, planeación, construcción y pruebas. EL punto crítico se encuentra en completar los servicios REST de las aplicaciones móviles así como del servidor puesto que son las que más trabajo representan además que deben tener uniformidad.

|  |  |  |
| --- | --- | --- |
|  |  | Tiempo estimado (Horas) |
| Planeación | Investigación técnicas balance carga | 1 |
| Asignación de tareas | 1 |
| Planeación | 3 |
| Desarrollo | Mobibus Stand Alone | 5 |
| Vcub Stand Alone | 5 |
| Tren Eléctrico Stand Alone | 5 |
| Rest aplicación | 10 |
| Implementación balanceo | 5 |
| Bono | 5 |
| Pruebas | Pruebas | 5 |
| Corregimiento | 10 |
| Pruebas | 5 |
| Otros | Instalación software | 10 |
| Desarrollo reporte | 4 |
|  | Total Estimado | 296 |

**Post experimentación**

Para satisfacer la disponibilidad del 100% se decidió utilizar balance de carga como se muestra en la imagen 1

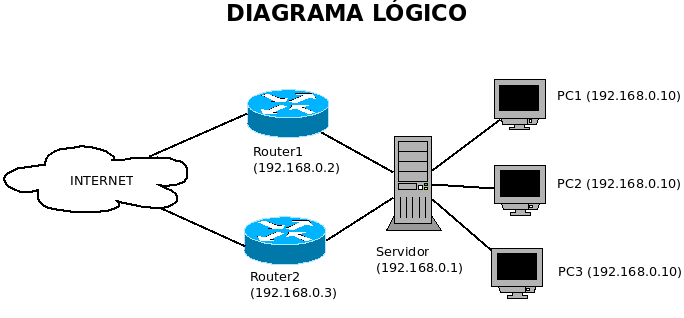


Imagen 1 Balanceo de carga

Balanceo de carga.

El balance de carga ha sido uno de los problemas principales de los sistemas de computación distribuidos. Esto, ya que se deben combinar especificaciones del cliente así como las restricciones del sistema para obtener la mejor respuesta posible. Adicionalmente el poder distribuir los recursos durante la ejecución de una solicitud para así asegurar que se responda lo antes posible sin consumir recursos extra. A este problema se le llama balanceo de carga dinámico.

Existen distintas técnicas de balanceo d carga que se aplican mejor a diferentes problemas. A continuación se presentan una lista de técnicas consideradas para la solución deseada. [1]

Round Robin: Distribuye cada petición en una línea de servidores para que así eventualmente cada servidor tenga la misma cantidad de conexiones que los demás. Se considera esta técnica puesto que funciona bien en la mayoría de los casos. Sin embargo existen peticiones que tiene una mayor prioridad que otras como por ejemplo la emergencia de un Mobibus.

Ratio (member) Ratio (node): Distribuye las peticiones en los servideros de acuerdo a una ponderación definida por el usuario. Esta técnica se considera ya que se tiene un equipo con más capacidad de procesamiento entonces este debería tener una mayor ponderación.

Dynamic Ratio (member) Dynamic Ratio (node): Ratio (member) Ratio (node) solo que las ponderaciones las hace el sistema automáticamente basándose en el comportamiento de los servidores. Esta técnica posee la ventaja de detectar posibles fallos en los servidores como pueden ser que un servidor se bloquee o que su conexión sea más lenta que la de los demás.

Least Connections (member) Least Connections (node): Distribuye las peticiones dependiendo de sus conexiones activas. Siendo a los servidores con menores conexiones a quienes se les pasa la nueva petición. Este método es más apropiado de utilizar cuando los servidores tienen capacidades de procesamiento similares, además se debe tener cuidado con la saturación de servidores ya que puede que un servidor tenga más conexiones tenga una capacidad mayor de procesamiento.

Weighted Least Connections (member) Weighted Least Connections (node): Distribuye las peticiones no solo dependiendo de las conexiones sino que además de la capacidad de procesamiento del servidor. Se utiliza como criterio de selección el % de capacidad que se calcula como las conexiones del servidor sobre su capacidad. La nueva petición se le asigna al servidor con menor % de capacidad. Este método sirve mejor para servidores con diferentes capacidades.

Predictive (member) Predictive (node): Observa el comportamiento de cada servidor y así saber cual tiene una mayor o menor rendimiento. Las nuevas peticiones se asignan a los servidores que tengan un mayor rendimiento. Esta técnica no es recomendada para grandes números de peticiones.

Se decidió probar con Round Robin y Ratio (member) Ratio (node) puesto que estos son los que más se ajustan a las características de los equipos con los que se cuenta. Es decir nodos con capacidades de procesamiento similares. Para esto se selección nginx como servidor por su facilidad de configuración. Además se utilizó el cache en las pruebas de carga para disminuir el tiempo de respuesta.

Resultados:

Comparación resultados con y sin balanceo de carga.

Para los resultados se utilizó el modelo de Ratio (member) Ratio (node) ya que Round Robin dio resultados pésimos. Además las preubas se vieorn sesgadas pro la cantidad de usuarios conectados a la red de la universidad. Esto trato de disminuirse al conectar los quipos involucrados al Ethernet pero los resultados fueron los mismos. Entonces las pruebas se hicieron cuando hubo una baja cantidad de usuarios en la red, en salones vacíos o durante la noche. Sin embargo, este factor muestra que las gráficas no son tan confiables y los resultados que se obtienen.

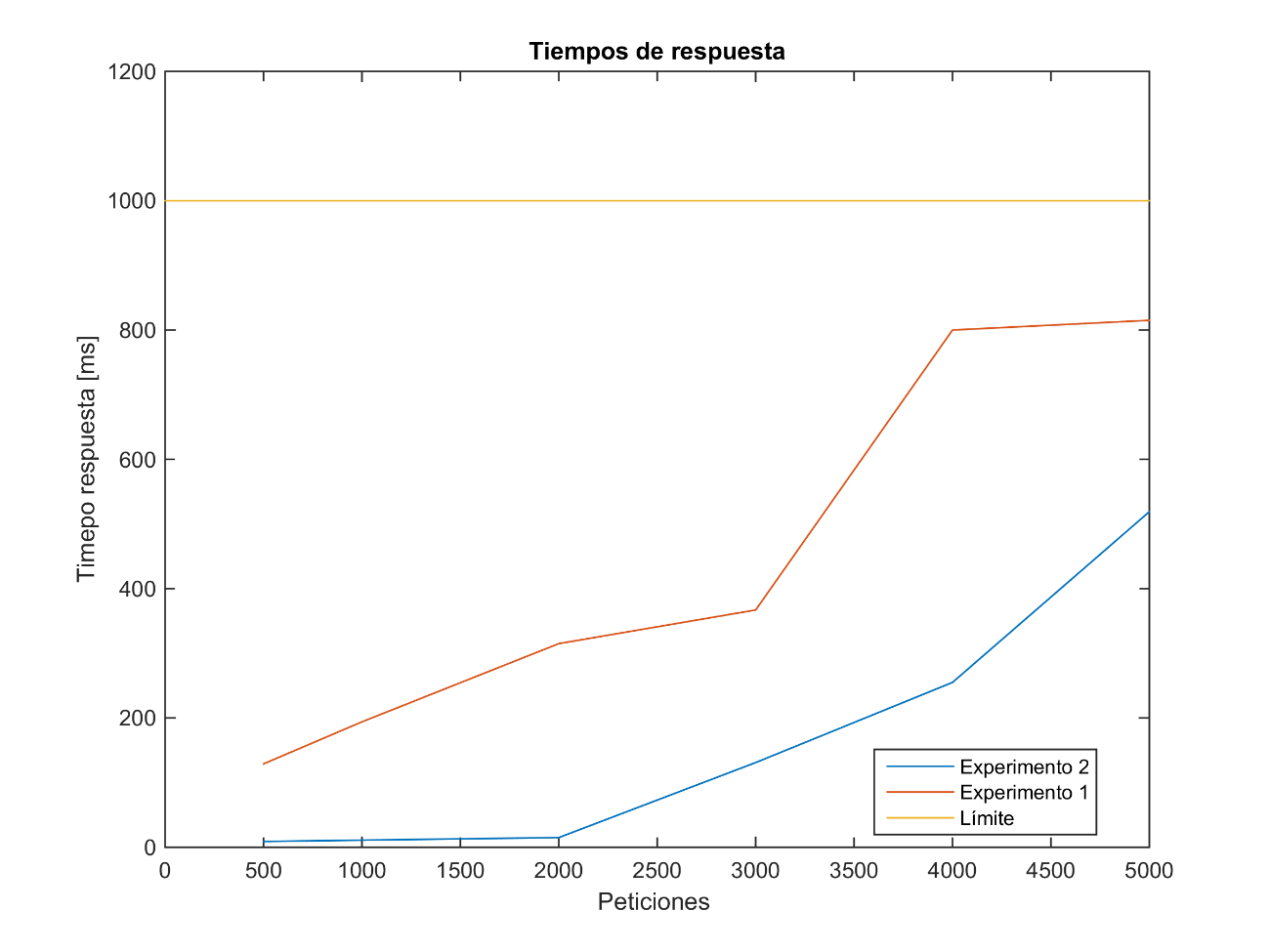


Ilustración Comparación tiempos de respuesta método GET entrega 1 vs entrega 2

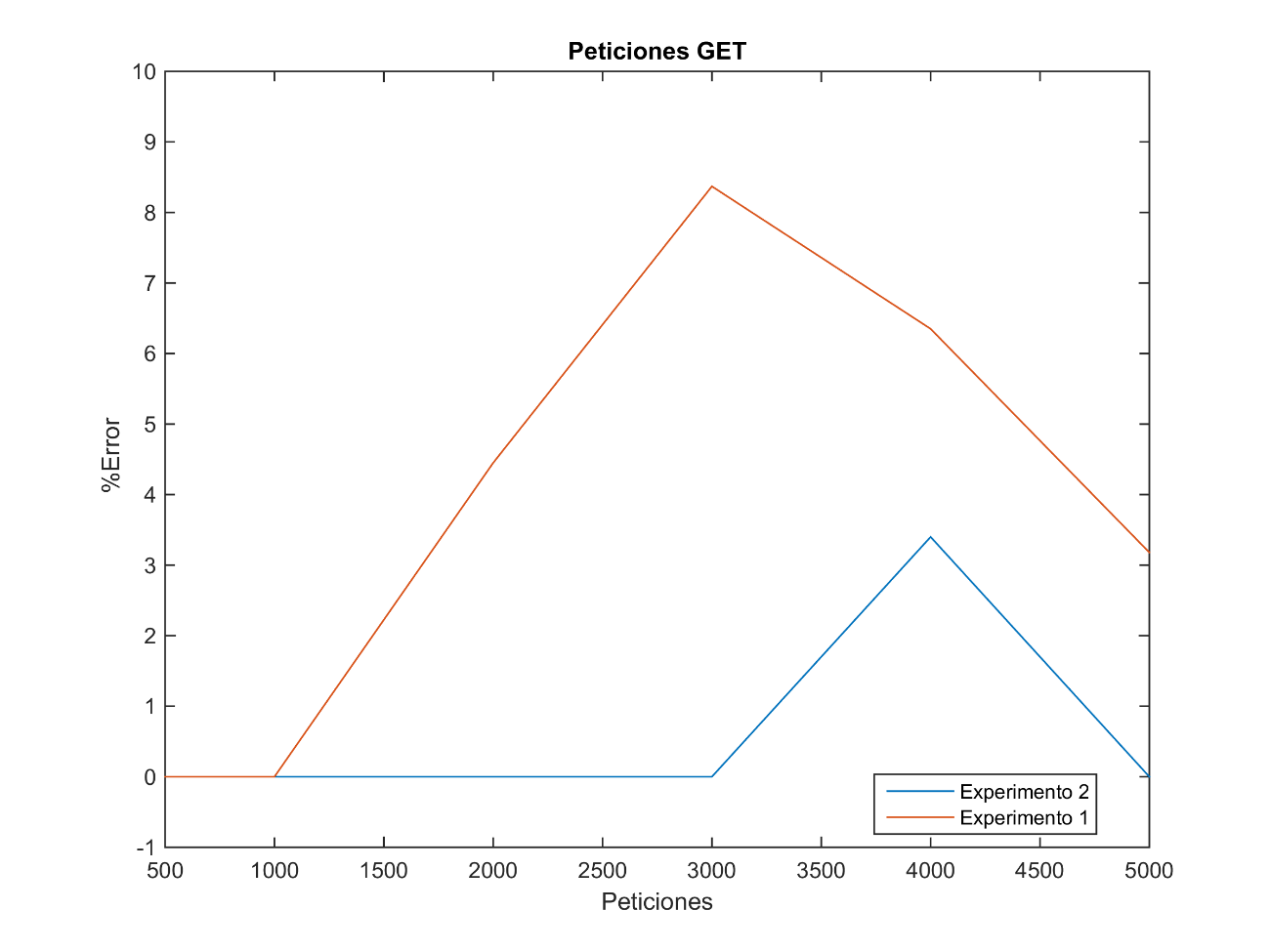


Ilustración Comparación % error obtenido entrega 1 vs entrega 2 método GET

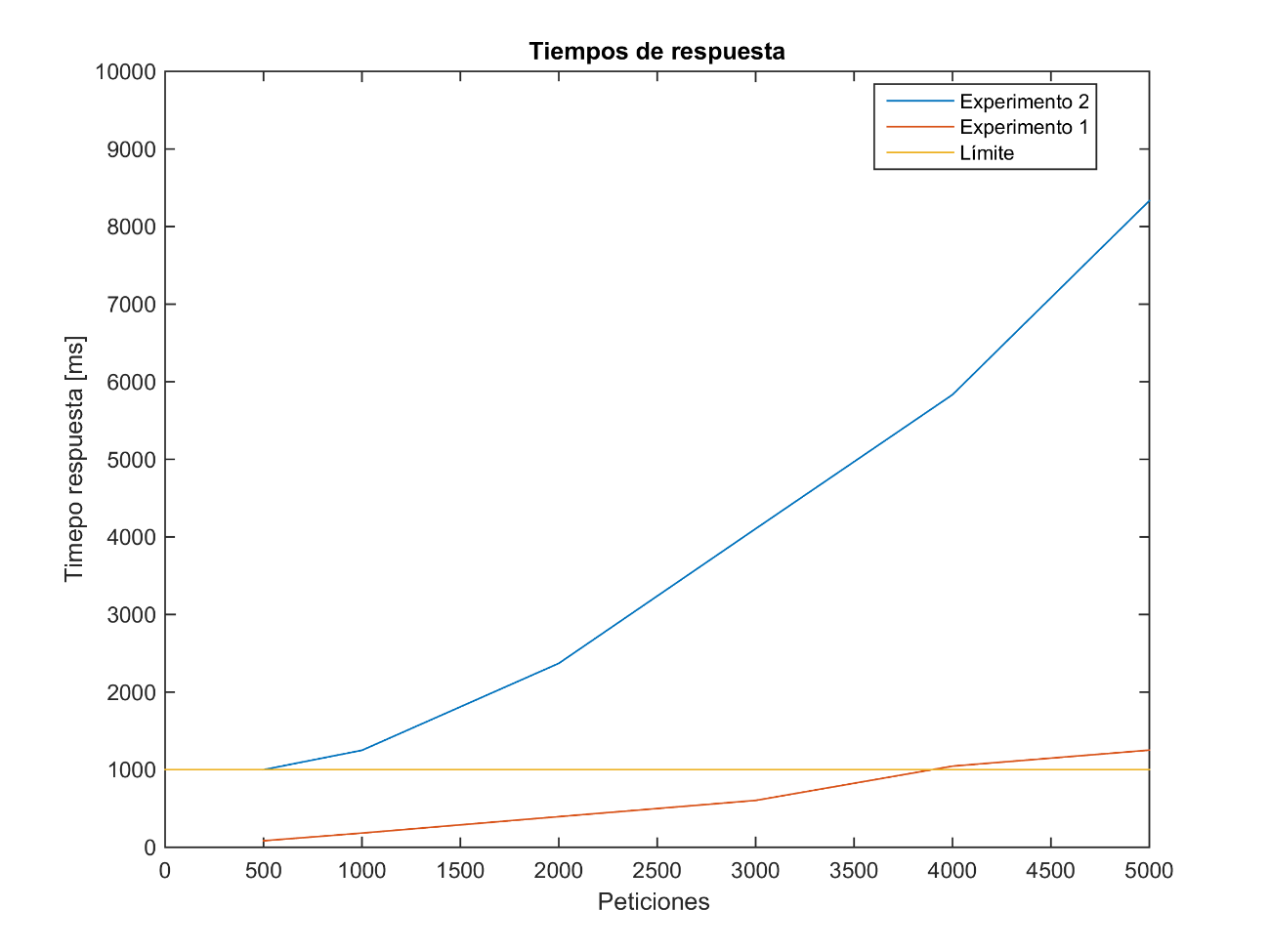


Ilustración Comparación tiempos de respuesta método POST entrega 1 vs entrega 2

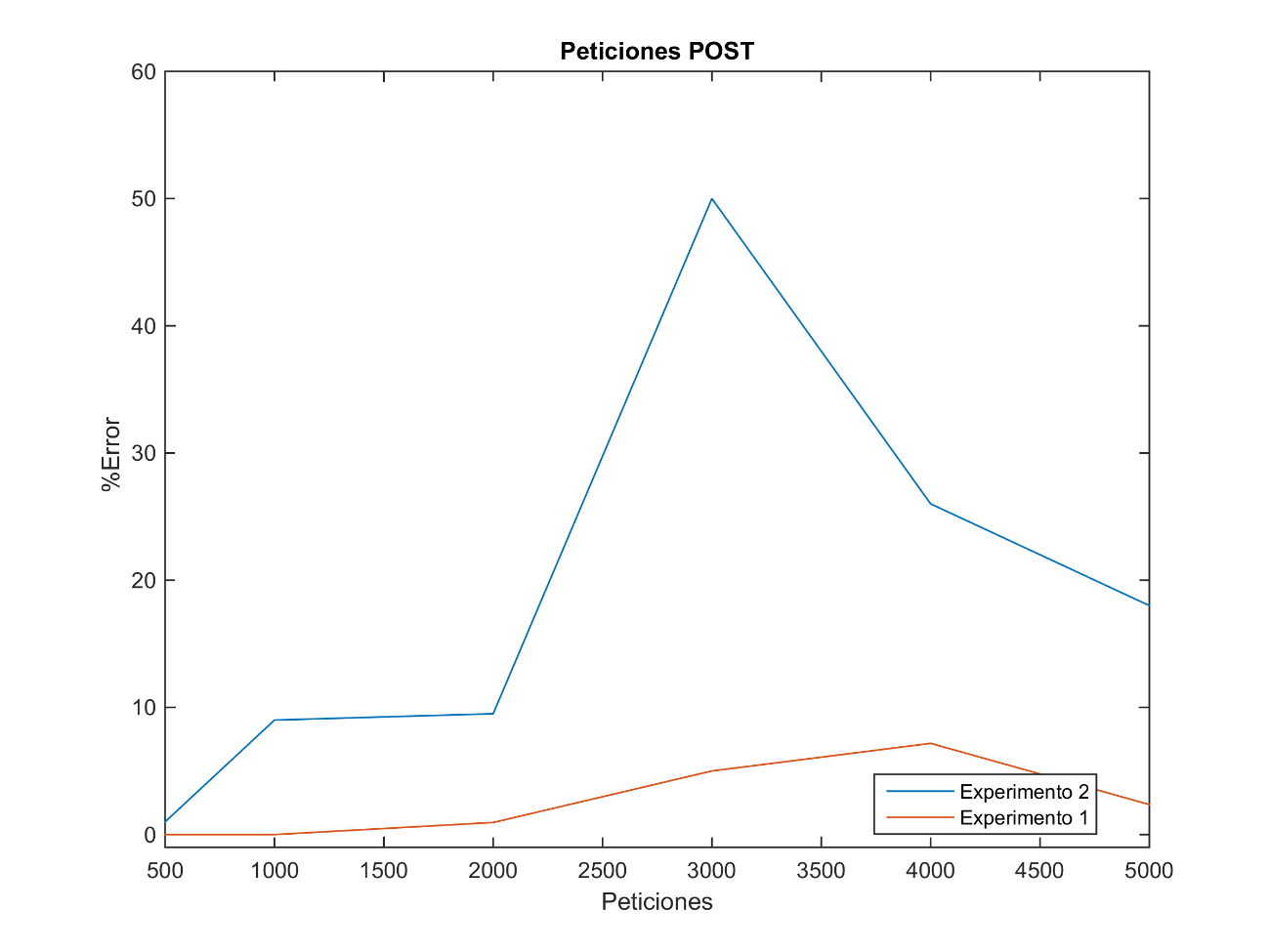


Ilustración Comparación % error obtenido entrega 1 vs entrega 2 método POST

Tiempos reales

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Tiempo estimado (Horas) | Jairo | Juan | Meili | Simón |
| Planeación | Investigación técnicas balance carga | 1 | 1 | 0 | 0 | 0 |
| Asignación de tareas | 1 | 1 | 1 | 0 | 1 |
| Planeación | 3 | 0 | 2 | 3 | 3 |
| Desarrollo | Mobibus Stand Alone | 5 | 4 | 1 | 4 | 0 |
| Vcub Stand Alone | 5 | 4 | 0 | 1 | 0 |
| Tren Eléctrico Stand Alone | 5 | 0 | 4 | 1 | 4 |
| REST aplicación | 10 | 6 | 4 | 5 | 4 |
| Implementación balanceo | 5 | 2 | 3 | 2 | 4 |
| Bono | 5 | 0 | 0 | 0 | 0 |
| Pruebas | Pruebas | 5 | 5 | 0 | 2 | 0 |
| Corregimiento | 10 | 2 | 7 | 6 | 6 |
| Pruebas | 5 | 2 | 3 | 4 | 5 |
| Otros | Instalación software | 10 | 3 | 3 | 3 | 3 |
| Desarrollo reporte | 4 | 3 | 1 | 3 | 2 |
|  | Total Estimado | 296 | 33 | 29 | 34 | 32 |
|  | Real | 138 |  |  |  |  |
|  | Diferencia | 53% |  |  |  |  |

Artefactos construidos

3 aplicaciones móviles: Mobibus, Vcub y tranvía eléctrico.

Servidor con base de datos en postgres SQL local. Todos los métodos deseados en la entrega 1. Archivo config de nginx que permite hacer el balance de carga a través del puerto 90.

Análisis

En general, se observó como el tiempo de latencia disminuyo drásticamente, así mismo los % de error. SIN embargo, le objetivo de esta entrega era el 0% de error y por ejemplo en al ilustración 2 se observa que se obtuvo un 3.5% de peticiones no cumplidas de las 3000 enviadas a pesar de que en otra prueba de 5000 peticiones 5000 se respondieron exitosamente. Se atribuye este comportamiento al hecho de que el servidor estaba en otro computador así como el tiempo de demora delay de la base de datos. Entonces, estas pruebas representan más le comportamiento que la aplicación tendrá en el ambiente fuera de la experimentación.

Se observa en las ilustraciones 3 y 4 que en cuanto al método POST no se cumple ninguno de los atributos de calidad propuesto , esto significa que el equipo debe plantearse invertir más en recursos físico o cambiar la lógica. Esto, a pesar de que se esté utilizando POSTGRES SQL así como pgbouncer para manejar sus peticiones.

A pesar de todo lo anterior se resalta que el balanceador de carga si está cumpliendo con sus tareas, puesto que al hacerlo todo con un solo equipo los errores y el tiempo de repsuesta se incrementan de manera significativa. Siendo los máximos de estos 70% y 20s respectivamente para 5000 usuarios.

Conclusiones

Falto trabajo en equipo así como el apoyo mutuo de los miembros del gripo. Además de motivación por el problema. Sine embargo, el resultado final fue satisfactorio y permitió ver el alcanza del balanceador de carga en un proyecto real. Además que se debe reconsiderar el mecanismo de sincronización de base (routing de base de datos de Django)de datos o incluir nuevos equipos a la red.