**Documento Parcial Experimento 2**

**Contenido:**

1. Atributos de calidad

* Desempeño
* Escalabilidad
* Disponibilidad
* Pre-Experimentación
* Experimentación
* Post-Experimentación

1. Resumen
2. Arquitectura Elegida
3. Comparación con entrega pasada 1.1-1.2
4. Comparación con entrega pasada 1.2-2.1
5. Conclusión

**Atributos de calidad**

Se van a probar únicamente dos escenarios de calidad que están en rojo.

* **Desempeño**

|  |  |
| --- | --- |
| **Identificador** | E-1.1 |
| **Prioridad** | Alta |
| **Fuente** | Cliente discapacitado. |
| **Estímulo** | Reservar mobibus (Transporte para discapacitados). |
| **Ambiente** | Normal |
| **Medida esperada** | Latencia menor a 1 segundo y 0% de error. |

|  |  |
| --- | --- |
| **Identificador** | E-1.2 |
| **Prioridad** | Alta |
| **Fuente** | Cliente discapacitado. |
| **Estímulo** | Cancelar reserva mobibus (Transporte para discapacitados). |
| **Ambiente** | Normal |
| **Medida esperada** | Latencia menor a 1 segundo y 0% de error. |

|  |  |
| --- | --- |
| **Identificador** | E-1.3 |
| **Prioridad** | Alta |
| **Fuente** | Cliente |
| **Estímulo** | Prestar Vcub (Bicicleta). |
| **Ambiente** | Normal |
| **Medida esperada** | Latencia menor a 1 segundo y 0% de error. |

|  |  |
| --- | --- |
| **Identificador** | E-1.4 |
| **Prioridad** | Media |
| **Fuente** | Cliente. |
| **Estímulo** | Devolver Vcub (Bicicleta). |
| **Ambiente** | Normal |
| **Medida esperada** | Latencia menor a 1 segundo y 0% de error. |

|  |  |
| --- | --- |
| **Identificador** | E-1.5 |
| **Prioridad** | Alta |
| **Fuente** | Encargado central de información. |
| **Estímulo** | Conocer ubicación vehículos. |
| **Ambiente** | Normal |
| **Medida esperada** | Latencia menor a 1 segundo y 0% de error. |

|  |  |
| --- | --- |
| **Identificador** | E-1.6 |
| **Prioridad** | Baja |
| **Fuente** | Encargado estación Vcubs. |
| **Estímulo** | Informar estación vacía de Vcub (Bicicletas). |
| **Ambiente** | Normal |
| **Medida esperada** | Latencia menor a 1 segundo y 0% de error. |

|  |  |
| --- | --- |
| **Identificador** | E-1.7 |
| **Prioridad** | Baja |
| **Fuente** | Encargado estación Vcubs. |
| **Estímulo** | Obtener disponibilidad estación de Vcub (Bicicletas). |
| **Ambiente** | Normal |
| **Medida esperada** | Latencia menor a 1 segundo y 0% de error. |

|  |  |
| --- | --- |
| **Identificador** | E-1.8 |
| **Prioridad** | Baja |
| **Fuente** | Encargado estación Vcubs. |
| **Estímulo** | Informar estación vacía de Vcub (Bicicletas). |
| **Ambiente** | Normal |
| **Medida esperada** | Latencia menor a 1 segundo y 0% de error. |

|  |  |
| --- | --- |
| **Identificador** | E-1.9 |
| **Prioridad** | Alta |
| **Fuente** | Encargado central de información. |
| **Estímulo** | Conocer el estado de un vehiculo. |
| **Ambiente** | Normal |
| **Medida esperada** | Latencia menor a 1 segundo y 0% de error. |

|  |  |
| --- | --- |
| **Identificador** | E-1.10 |
| **Prioridad** | Baja |
| **Fuente** | Encargado central de información. |
| **Estímulo** | Agregar reporte de un vehículo. |
| **Ambiente** | Normal |
| **Medida esperada** | Latencia menor a 1 segundo y 0% de error. |

|  |  |
| --- | --- |
| **Identificador** | E-1.11 |
| **Prioridad** | Baja |
| **Fuente** | Encargado estación Vcubs. |
| **Estímulo** | Registrar Vcub (Bicicletas). |
| **Ambiente** | Sobrecarga |
| **Medida esperada** | Latencia menor a 1 segundo y 0% de error. |

|  |  |
| --- | --- |
| **Identificador** | E-1.12 |
| **Prioridad** | Alta |
| **Fuente** | Encargado central de información. |
| **Estímulo** | Reportar emergencia. |
| **Ambiente** | Sobrecarga |
| **Medida esperada** | Latencia menor a 1 segundo y 0% de error. |

|  |  |
| --- | --- |
| **Identificador** | E-1.13 |
| **Prioridad** | Baja |
| **Fuente** | Encargado central de información. |
| **Estímulo** | Conocer información de conductores. |
| **Ambiente** | Sobrecarga |
| **Medida esperada** | Latencia menor a 1 segundo y 0% de error. |

* **Escalabilidad**

|  |  |
| --- | --- |
| **Identificador** | E-2.14 |
| **Prioridad** | Alta |
| **Fuente** | GPS de vehículos. |
| **Estímulo** | Recibir posición de un vehículo. |
| **Ambiente** | Sobrecarga |
| **Medida esperada** | Poder atender 4500 vehículos en 5 segundos. |

* **Disponibilidad**
* **Pre-Experimentación**
  + Problemática: La arquitectura definida podrá cumplir con los requerimientos no funcionales que se plantearon para la entrega. En primer lugar, que la latencia de las funcionalidades no exceda 1 segundo. Por otro lado, que el sistema soporte hasta 4500 vehículos enviando información al servidor en 5 segundos.
  + Objetivo del experimento: Probar los requerimientos no funcionales con el backend implementado y verificar si se cumplen las métricas establecidas.
  + Descripción del experimento: Se deben hacer dos tipos de pruebas, esto dependerá si para ese servicio se va a probar la escalabilidad o el desempeño. Se utilizarán pruebas de carga en las que se incrementa progresivamente la cantidad de peticiones.
  + Artefactos a construir: Se debe haber desarrollado toda la lógica de los servicios que se vayan a probar.
  + Recursos de la experimentación:

*Software*:

*Lenguajes*: Java, Scala

*Frameworks*: Play!, persistencia JPA

*Hardware*:

*Procesador*: Intel(R) Xeon(R) CPU ES-2665 0 @ 2.40 GHz( 2 procesadores)

*Memoria instalada* (RAM)= 8,00 GB

*Tipo de sistema*: Sistema operativo de 64 bits

* + Resultados esperados:

Los resultados esperados, son mencionados en el punto anterior respecto a cada atributo de calidad. En donde, se especifica el atributo con un respectivo escenario y el estímulo.

* **Experimentación:**

Desempeño:

E-1.1: Reservar mobibus

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***# de usuarios*** | ***Media (ms)*** | ***Min (ms)*** | ***Max (ms)*** | ***% error*** | ***Rendimiento (peticiones/segundo)*** |
| 10 | 101 | 55 | 183 | 0 | 188,7 |
| 100 | 662 | 203 | 977 | 0 | 280,9 |
| 500 | 374 | 15 | 925 | 0 | 528,0 |
| 1000 | 547 | 2 | 1527 | 0 | 821,5 |
| 1500 | 423 | 1 | 1271 | 0 | 996,0 |
| 2000 | 743 | 2 | 1788 | 0 | 1207,7 |
| 2500 | 532 | 2 | 1516 | 0 | 1336,1 |
| 3000 | 663 | 3 | 1853 | 0 | 189,7 |
| 3500 | 532 | 2 | 1902 | 0 | 972,0 |
| 4000 | 587 | 2 | 2289 | 0 | 1336 |

E-1.2: Cancelar reserva

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***# de usuarios*** | ***Media (ms)*** | ***Min (ms)*** | ***Max (ms)*** | ***% error*** | ***Rendimiento (peticiones/segundo)*** |
| 4000 | 86 | 2 | 487 | 0 | 370 |

E-1.3: Prestar Vcub

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***# de usuarios*** | ***Media (ms)*** | ***Min (ms)*** | ***Max (ms)*** | ***% error*** | ***Rendimiento (peticiones/segundo)*** |
| 10 | 20 | 6 | 126 | 0 | 2,2 |
| 100 | 7 | 5 | 126 | 0 | 49,5 |
| 500 | 3 | 2 | 18 | 0 | 97,3 |
| 1000 | 10 | 2 | 547 | 0 | 188,5 |
| 1500 | 3 | 1 | 113 | 0 | 285,7 |
| 1800 | 9 | 1 | 500 | 0 | 341,6 |
| 2000 | 2 | 1 | 251 | 0 | 361,6 |
| 3000 | 87 | 1 | 597 | 0 | 537,7 |
| 3500 | 89 | 1 | 1002 | 0 | 527,7 |
| 4000 | - | - | - | - | - |

E-1.4: Devolver Vcub

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***# de usuarios*** | ***Media (ms)*** | ***Min (ms)*** | ***Max (ms)*** | ***% error*** | ***Rendimiento (peticiones/segundo)*** |
| 10 | 9 | 8 | 11 | 0 | 2,2 |
| 100 | 7 | 6 | 15 | 0 | 20,1 |
| 500 | 25 | 3 | 441 | 0 | 96,6 |
| 1000 | 23 | 4 | 213 | 0 | 188,5 |
| 1500 | 56 | 2 | 703 | 0 | 285,7 |
| 1800 | 71 | 2 | 761 | 0 | 341,6 |
| 1900 | 1007 | 2 | 3793 | 0 | 351 |
| 2000 | 515 | 4 | 3099 | 0 | 361,6 |

E-1.5: Dar ubicación vehiculos

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***# de usuarios*** | ***Media (ms)*** | ***Min (ms)*** | ***Max (ms)*** | ***% error*** | ***Rendimiento (peticiones/segundo)*** |
| 4000 | 625 | 12 | 2369 | 0 | 470.9 |

E-1.6: Informar estacion vacia

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***# de usuarios*** | ***Media (ms)*** | ***Min (ms)*** | ***Max (ms)*** | ***% error*** | ***Rendimiento (peticiones/segundo)*** |
| 4000 | 125 | 6 | 1031 | 0 | 697.3 |

*E-1.7: Obtener disponibilidad estación de Vcub*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***# de usuarios*** | ***Media (ms)*** | ***Min (ms)*** | ***Max (ms)*** | ***% error*** | ***Rendimiento (peticiones/segundo)*** |
| 10 | 59 | 26 | 85 | 0 | 113,7 |
| 100 | 225 | 30 | 550 | 0 | 150,6 |
| 500 | 144 | 1 | 604 | 0 | 769.2 |
| 1000 | 313 | 2 | 772 | 0 | 697.8 |
| 1500 | 446 | 2 | 1246 | 0 | 670,8 |
| 2000 | 257 | 1 | 980 | 0 | 1109,4 |
| 2500 | 181 | 0 | 1033 | 0 | 898,3 |
| 3000 | 501 | 0 | 1481 | 0 | 1201,0 |
| 3500 | 623 | 1 | 2562 | 0 | 1245,6 |

E-1.9: Conocer el estado de un vehiculo

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***# de usuarios*** | ***Media (ms)*** | ***Min (ms)*** | ***Max (ms)*** | ***% error*** | ***Rendimiento (peticiones/segundo)*** |
| 4500 | 218 | 3 | 962 | 0 | 603.13 |

E-1.10: Agregar reporte

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***# de usuarios*** | ***Media (ms)*** | ***Min (ms)*** | ***Max (ms)*** | ***% error*** | ***Rendimiento (peticiones/segundo)*** |
| 4000 | 596 | 36 | 1839 | 0 | 505.1 |

E-1.11: Registrar vcub

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***# de usuarios*** | ***Media (ms)*** | ***Min (ms)*** | ***Max (ms)*** | ***% error*** | ***Rendimiento (peticiones/segundo)*** |
| 4000 | 456 | 1 | 6135 | 0 | 382.6 |

E-1.12: Reportar posición

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***# de usuarios*** | ***Media (ms)*** | ***Min (ms)*** | ***Max (ms)*** | ***% error*** | ***Rendimiento (peticiones/segundo)*** |
| 4500 | 392 | 1 | 1676 | 0 | 1399,6 |

E-2.14: Reportar emergencia

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***# de usuarios*** | ***Media (ms)*** | ***Min (ms)*** | ***Max (ms)*** | ***% error*** | ***Rendimiento (peticiones/segundo)*** |
| 4500 | 294 | 1 | 1392 | 0 | 1491,1 |

**Resumen**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Escenario de calidad** | **Atributo de calidad** | **Métrica** | **Valor esperado** | **Valor obtenido** |
| E-1.1 al E-1.13 | Desempeño | latencia | Menos de 1 segundo | 453.5 en promedio |
| E-1.1 al E-1.13 | Desempeño | %error | 0% | 0% |
| E-2.14 | Escalabilidad | rendimiento | 900 | 1491,1 |

**Arquitectura Elegida 1.2**

Play! es un framework que provee un mínimo uso de recursos (CPU, memoria, threads) aumentando la escalabilidad de la aplicación. Por otro lado, Play! utiliza un modelo MVC completamente asincrónico y sin estado (Arquitectura nada compartido) que juntos permite mantener tiempos reales de respuesta.

Con lo anterior se puede saber que se utiliza un patrón arquitectural basado en actores. La anterior consiste en un consumidor y productor que actúan según la petición REST que se realice. Además, para el manejo de los dos anteriores se cuenta con un supervisor quien es el encargado de balancear la carga y por ende mejorar el desempeño y la escalabilidad de esta arquitectura.

Mediante unas configuraciones hechas al proyecto en Play!, logramos controlar la capacidad del pool de threads que la aplicación ofrece y la volvimos puramente asincrónica. Mediante esta configuración, logramos que la escalabilidad aumentara puesto que se podían responder un mayor número de peticiones en cierto rango de tiempo.

Las bases de datos no relacionales manejan una estructura sencilla que permite una escalabilidad barata y sencilla, pero que sacrifica la integridad de los datos. Por otra parte, las bases de datos relacionales tienen una estructura jerárquica, compleja y normalizada que hace que escalar sea caro y complejo. No obstante, esta estructuración favorece las propiedades ACID (atomicidad, aislamiento, consistencia y durabilidad) de la información, y además permite generar servicios de consultas que en las no relacionales no se logran hacer tan sencillamente.

Debido a lo anterior, nosotros elegimos utilizar una base de datos relacional puesto que de esta manera lograríamos las propiedades ACID y por ende una mayor integridad de los datos que asegurarían unos buenos resultados en los servicios REST ofrecidos por nuestra aplicación. Además, mediante este tipo de base de datos lograríamos unos mejores tiempos de respuesta debido a los servicios de consulta logrados gracias a la estructuración de la información en la misma. Por último, cabe resaltar que esta decisión llevo a realizar un “trade-off” a favor del desempeño de la aplicación, y en contra de la escalabilidad de la misma.

Por último, se decidió desplegar la aplicación en la nube ya que al subirlo hay una mejora en la escalabilidad de la aplicación. Además, gracias a dicho despliegue se posibilitó dejar de utilizar una base de datos local que causaba una perdida en el desempeño de la aplicación. En la nube se logró utilizar una base de datos de PostgreSQL. Nuevamente se trata un “trade-off” entre desempeño y escalabilidad a la hora de tomar esta decisión, pero se llegó a la conclusión de que realizar el cambio tendría un mayor beneficio para la aplicación pues ofrecería una mayor escalabilidad, conectividad, elasticidad y una mejora en cuanto al desempeño de la base de datos de la aplicación.

RESPECTO A DISPONIBILIDAD

**Comparación con entrega pasada 1.1-1.2**

Para la entrega definitiva de este experimento hubo varios cambios importantes. En primer lugar, se realizó la capa de persistencia en una base de datos relacional pensando en lo explicado en el punto anterior. Por otra parte, se generó la aplicación standalone que permite simular el envió de posiciones de los carros que se encuentre registrados en la capa de persistencia anteriormente mencionada, mediante el consumo de los servicios REST generados para la entrega anterior. Además, se creó una capa de presentación web que permite utilizar los servicios REST de nuestra aplicación desde un navegador de internet. Por último, se subió y se desplego tanto la capa lógica como la capa de persistencia en la nube.

A lo largo de la experimentación, y en especial de la entrega pasada a esta entrega definitiva se realizaron diversos trade-off entre el desempeño y escalabilidad de la aplicación como fue explicado en las decisiones tomadas para nuestra arquitectura. Estos mismos se pueden ver reflejados en nuestros nuevos resultados. Además, el haber desplegado la base de datos en la nube influyo en las pruebas puesto que se realizaron teniendo datos en la misma, lo que dificulta cada petición. Como es posible ver, en el caso de los servicios POST el % de error, la latencia y el rendimiento empeoro.

Lo anterior se debe a que ahora los servicios están desplegados en la nube y con la base de datos en la misma, y debido a que se están utilizando opciones gratuitas de las mismas que no funcionan de la manera más óptima posible. Por dicha razón, el realizar dicho estilo de petición que exige mandar parámetros se volvió más costoso puesto que paso de ser local a ser en una instancia remota lejana. Por otra parte, los servicios PUT & DELETE también se demoran más debido a que las pruebas realizadas por JMeter intentan actualizar/borrar el mismo objeto tantas veces como peticiones se le ponga, lo que nos deja una gran problemática puesto que al intentar todos interactuar con un mismo objeto se vuelve un servicio plenamente sincrónico que genera una cola en la base de datos y por ende unos resultados malos. Por dicha razón, se concluyó que para este estilo de servicios hay que buscar otro estilo de prueba que si permita analizar bien el funcionamiento de la aplicación.

Por último, en cuanto a los servicios GET se puede ver que se mantiene el 0% de error y que el rendimiento y la latencia en gran medida mejoran o se mantienen. Lo anterior se debe, a que este es el servicio de menos exigencia y que por ende permite que incluso desplegando la aplicación en una entidad remota lejana se mantengan los resultados.

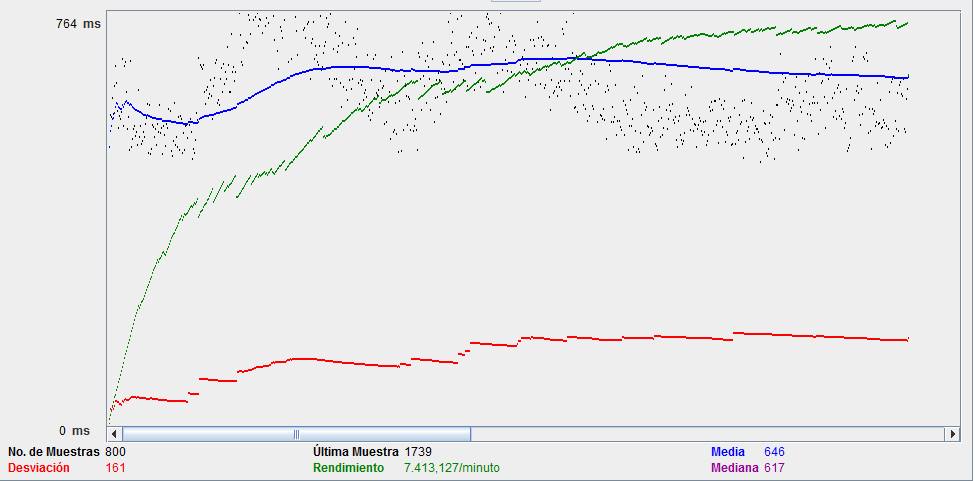
**Conclusiones 1.2**

Se llega a la conclusión de que es importante realizar tácticas y patrones arquitecturales ya que permiten el mejoramiento del software, tal como nos lo permitió las diversas decisiones arquitecturales tomadas a lo largo de este experimento. Además, se vio la importancia y el gran impacto de los famosos “trade-off” que se llevan al cabo a la hora de realizar una decisión arquitectural. Se comprendió que lo anterior se debe a que a la hora de tomar una decisión arquitectural hay que realizar cambios que no pueden favorecer a todos atributos de calidad, por lo que se concluyó que es de gran importancia hablar y/o negociar con los stakeholders de manera clara y concisa con el fin de poder determinar un orden de prioridad entre los requerimientos no funcionales para poder así tomar las decisiones arquitecturales de manera más eficiente y óptima. Por último, se llegó a resaltar la importancia de un arquitecto de software puesto que las decisiones del mismo pueden llegar a determinar si un proyecto permanecerá a flote o no debido a su gran impacto sobre los diversos escenarios de calidad que la aplicación deba llegar a cumplir.

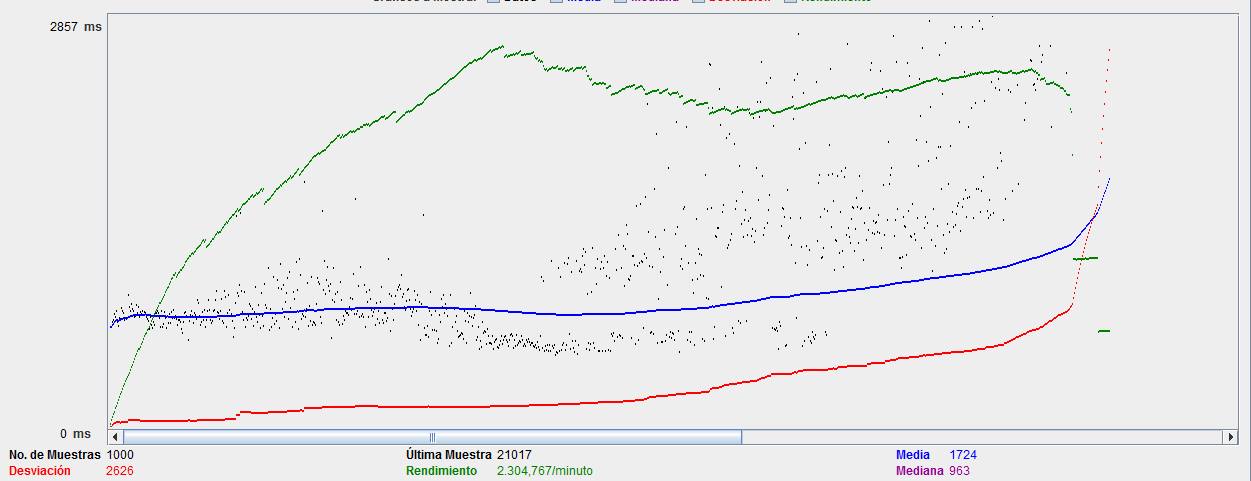
**PRUEBAS PASADAS**

*Pruebas post*

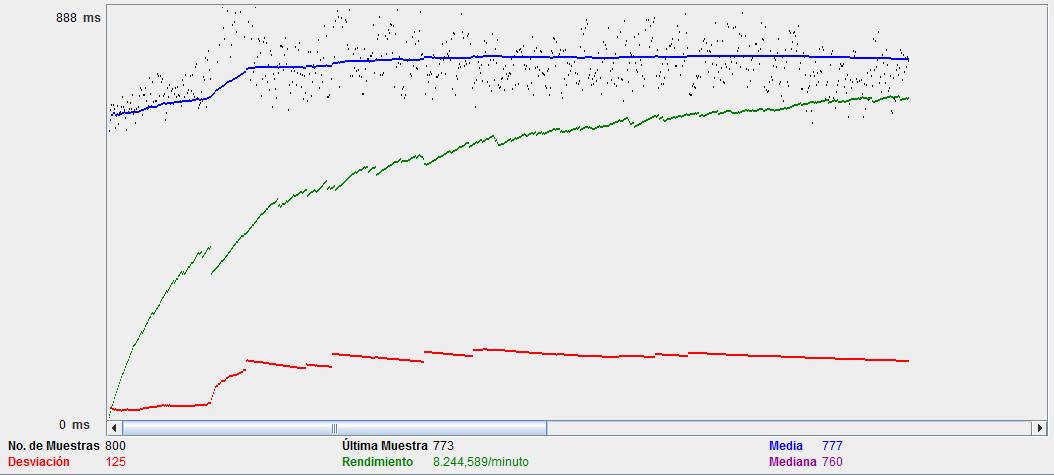
Post Informes.

Numero de Threads: 800

Numero de Threads: 1000



Post Vehiculo:

Numero de threads 800:

Post estaciones:

Numero de Threads 500:



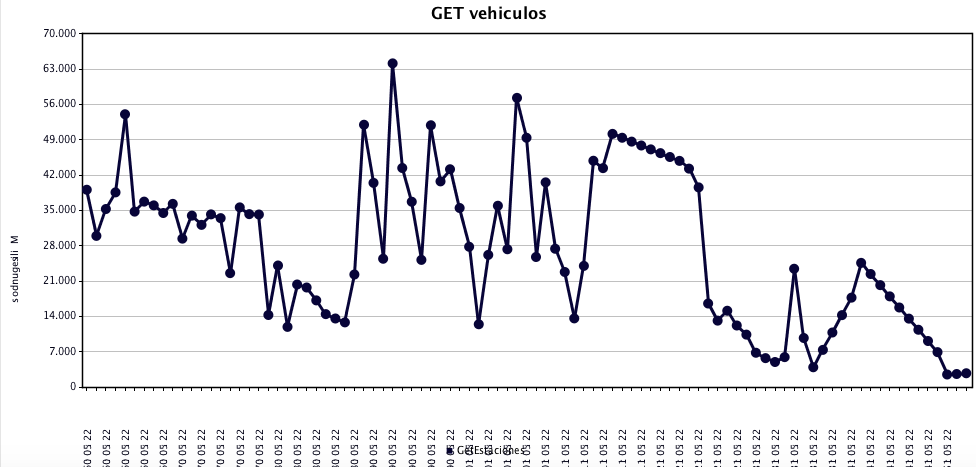
Numero de Threads 600:



Numero de Threads 800:

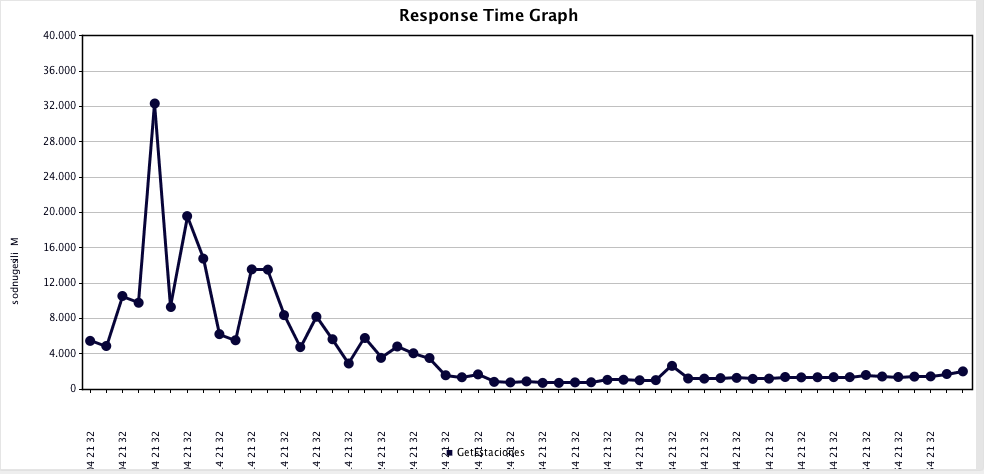


Get vehiculos



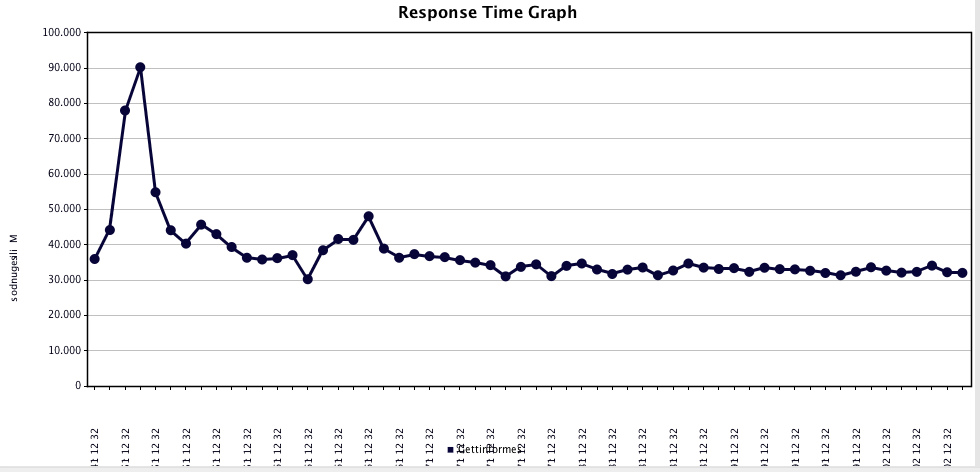


GET Estaciones

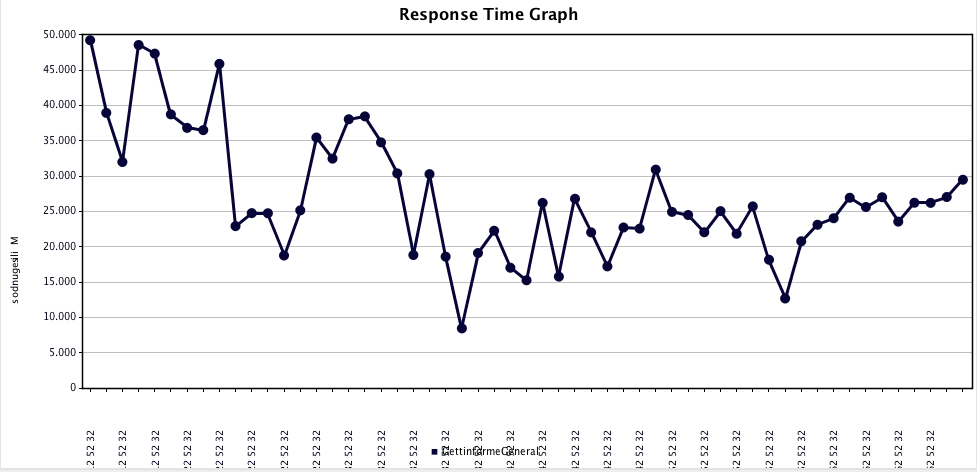




GET Informes



GET informe general



**PRUEBAS EXPERIMENTO 2**

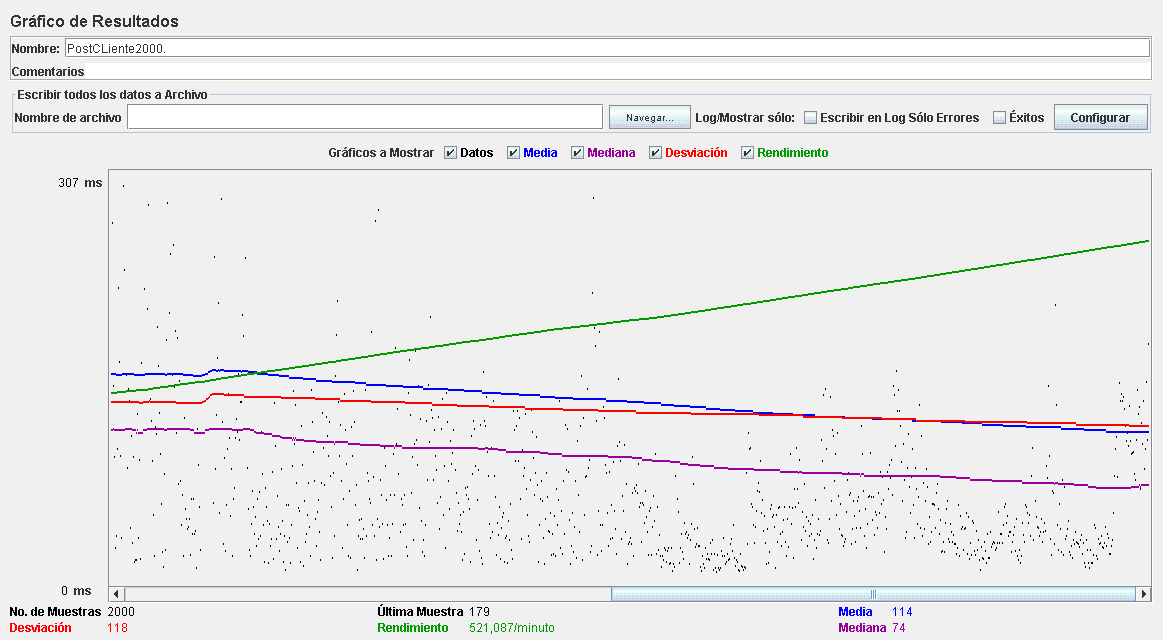
***Pruebas post***

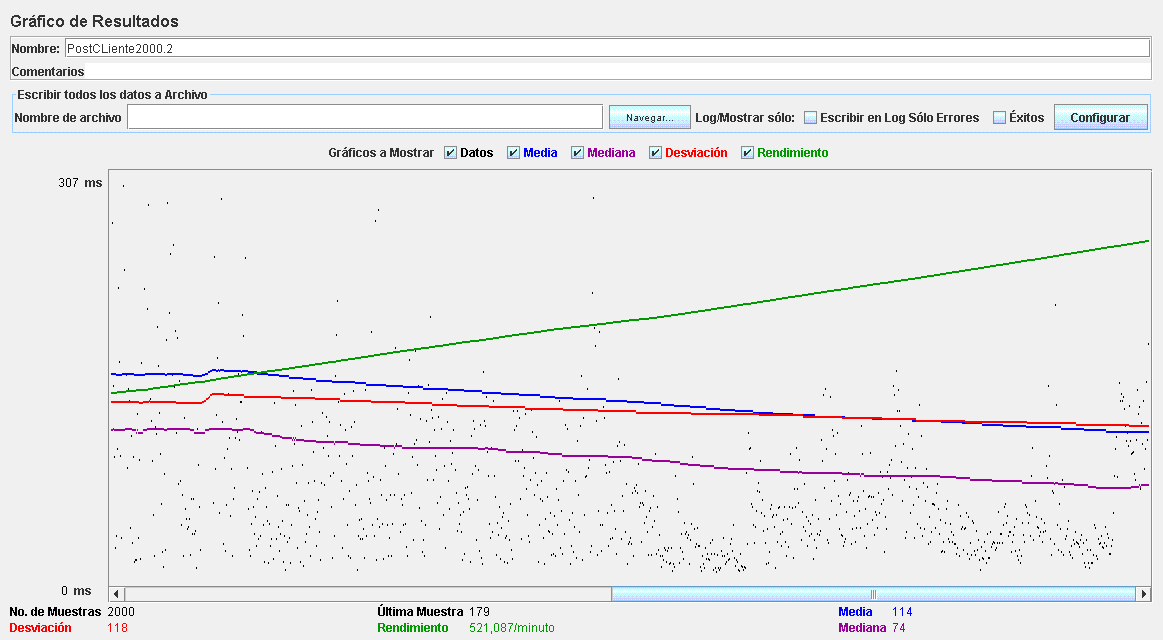
***Dos maquinas***

postCLiente

2000

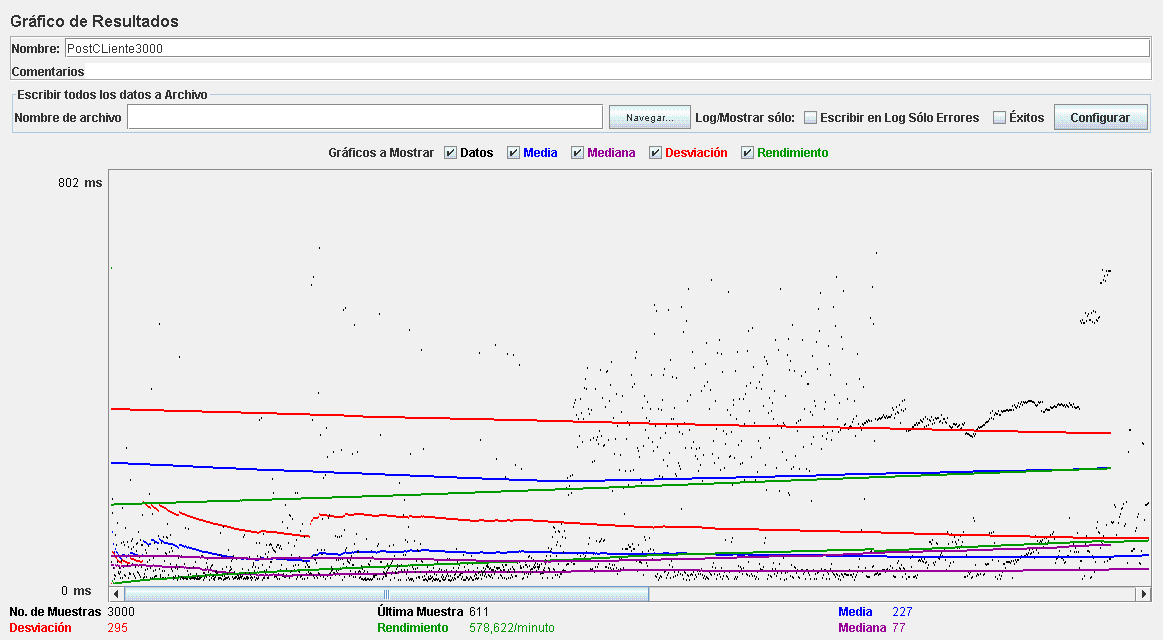
C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PostCliente2000.png

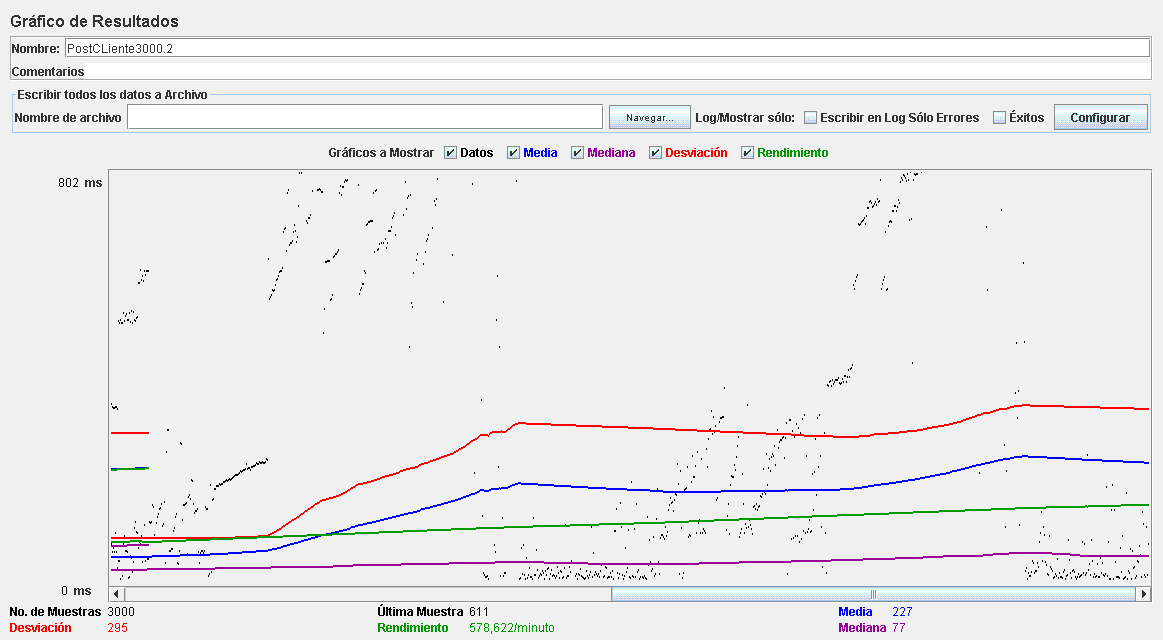




3000

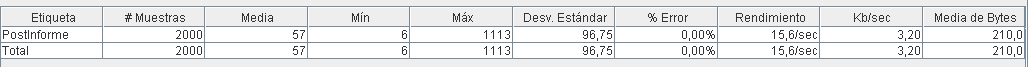
C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PostCliente3000.png

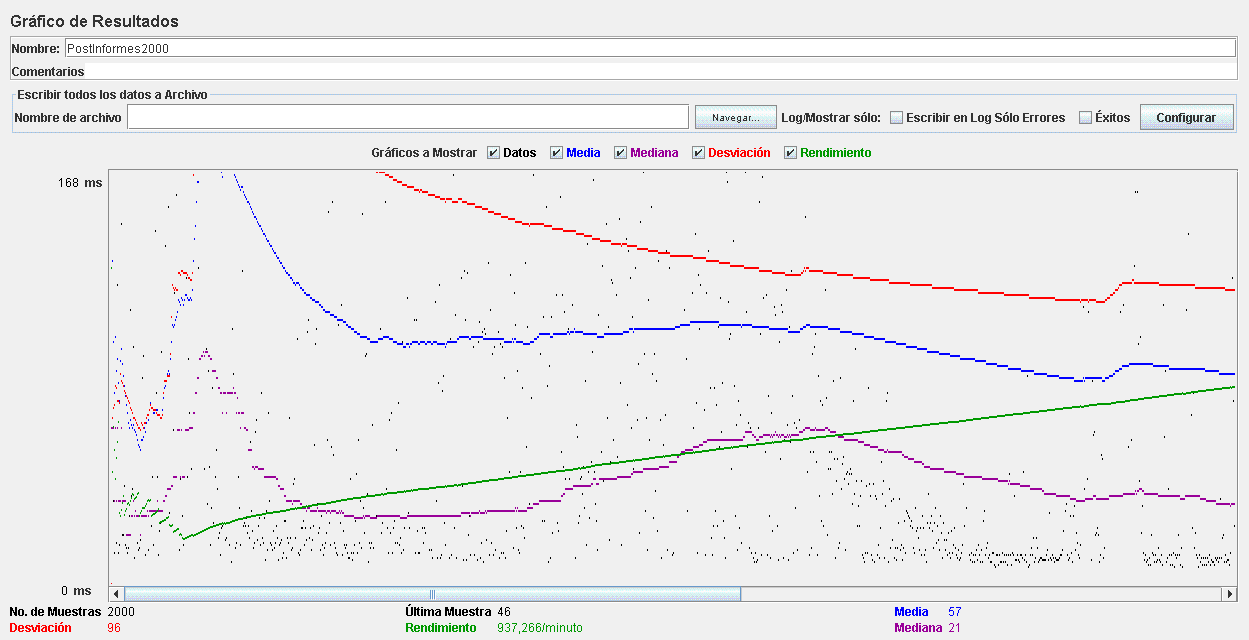


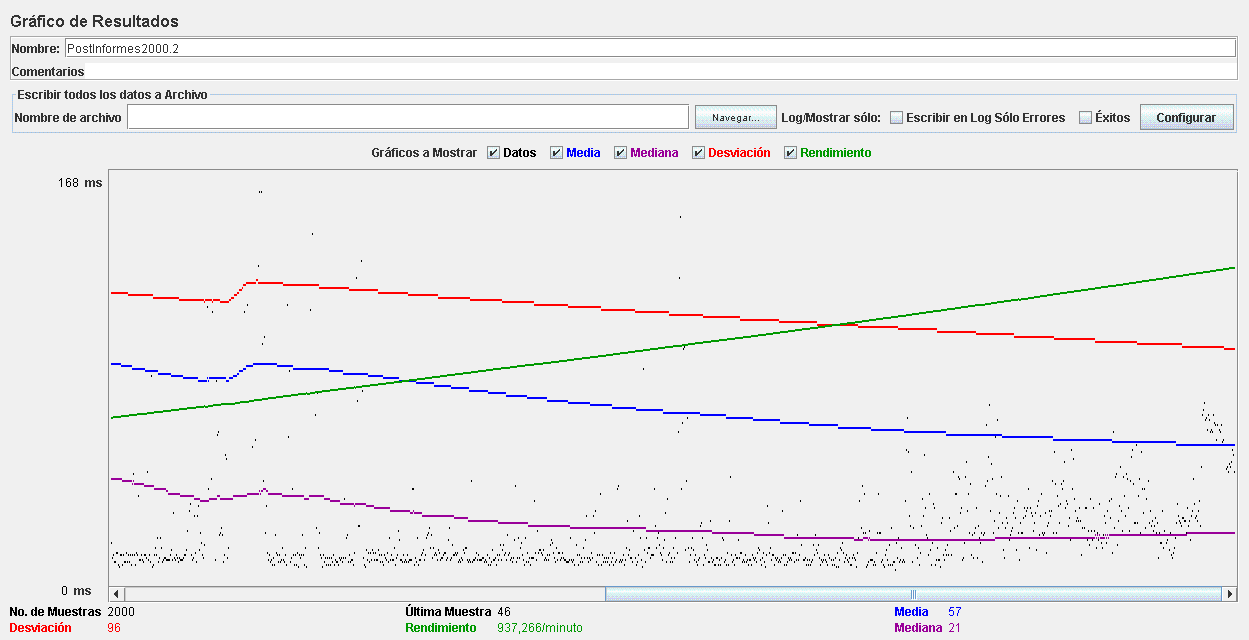


Post Informes.

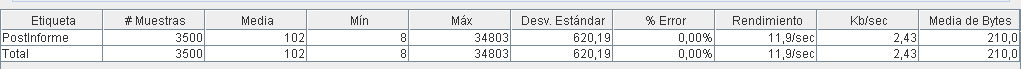
2000

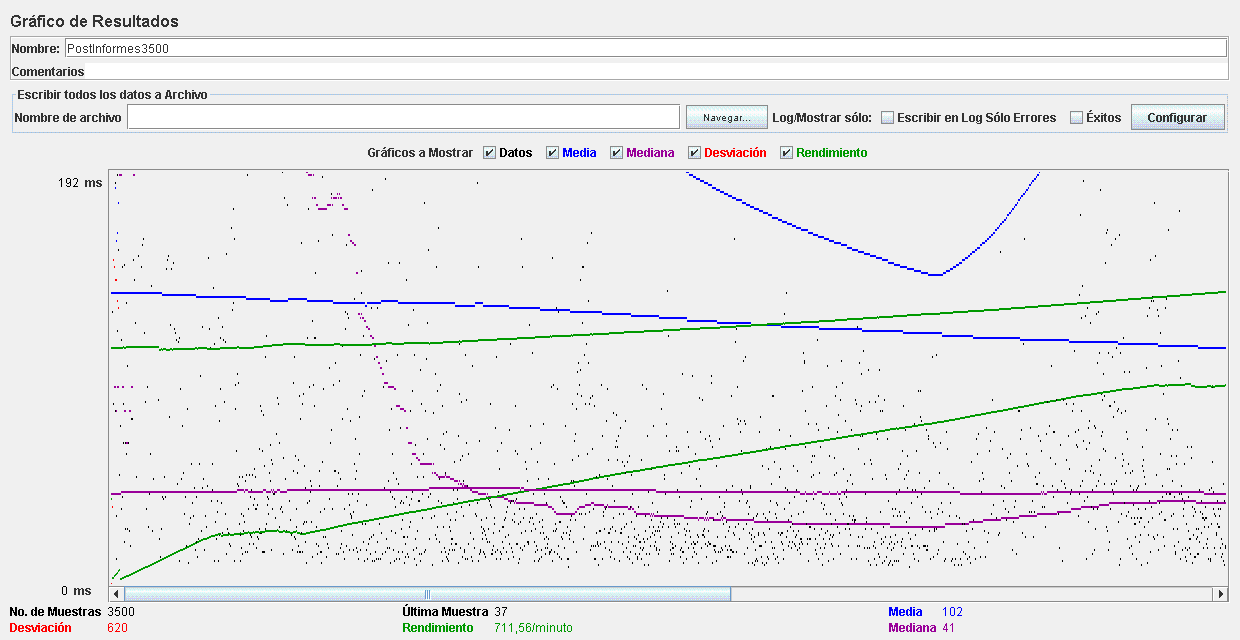


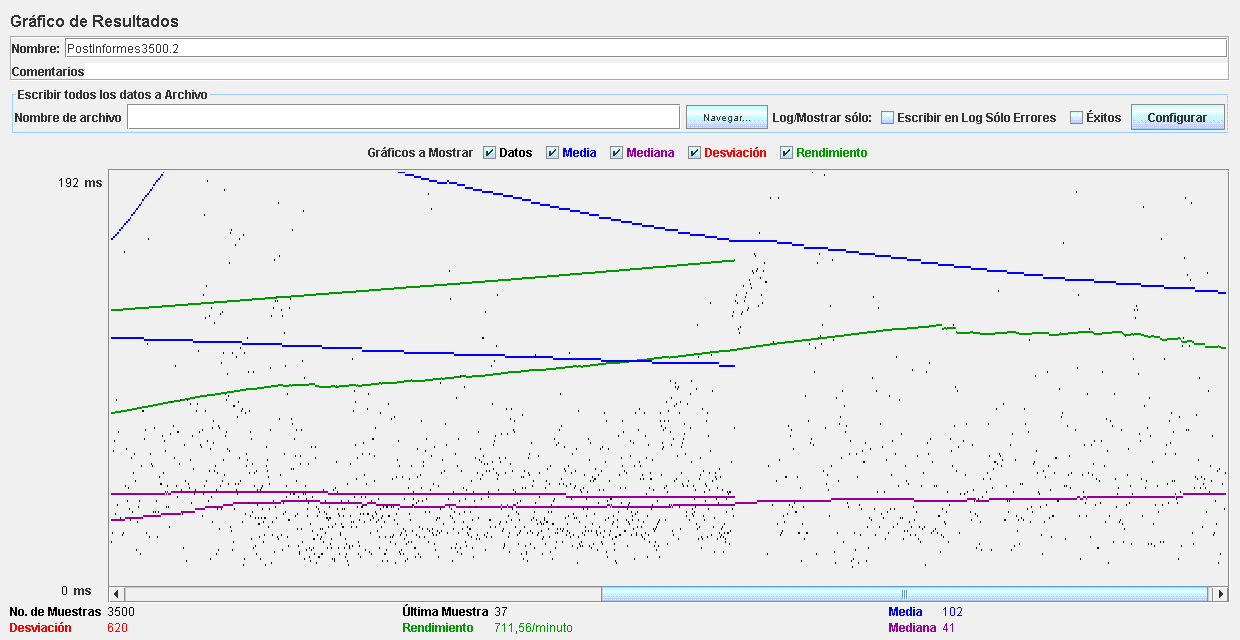




3500



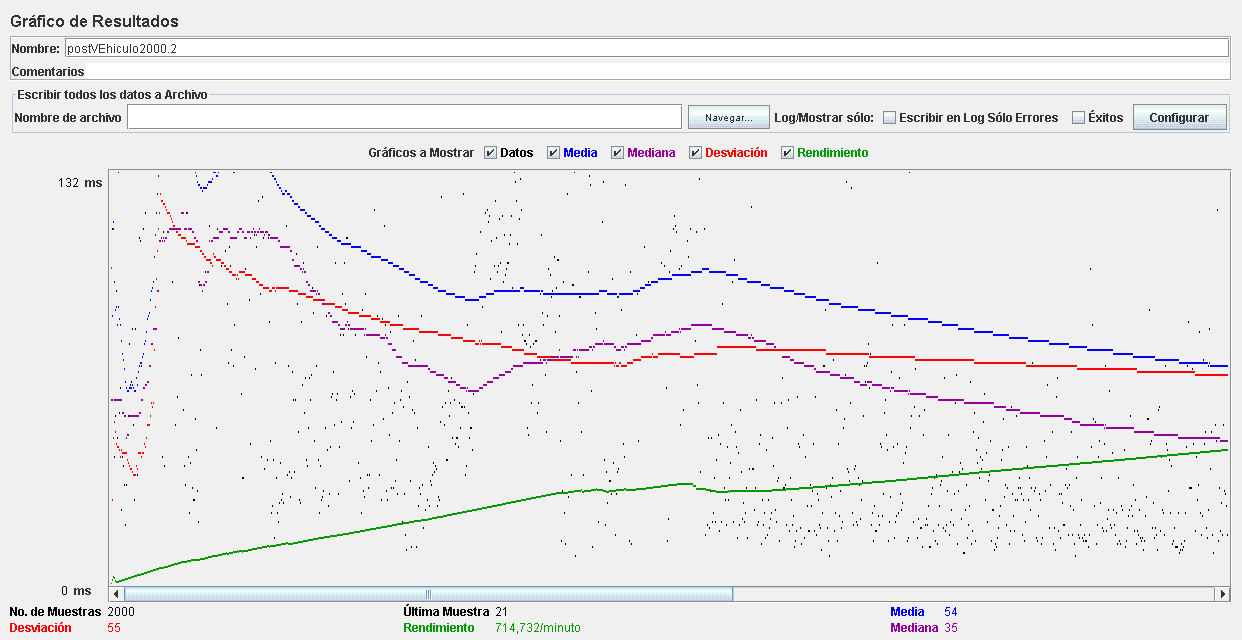


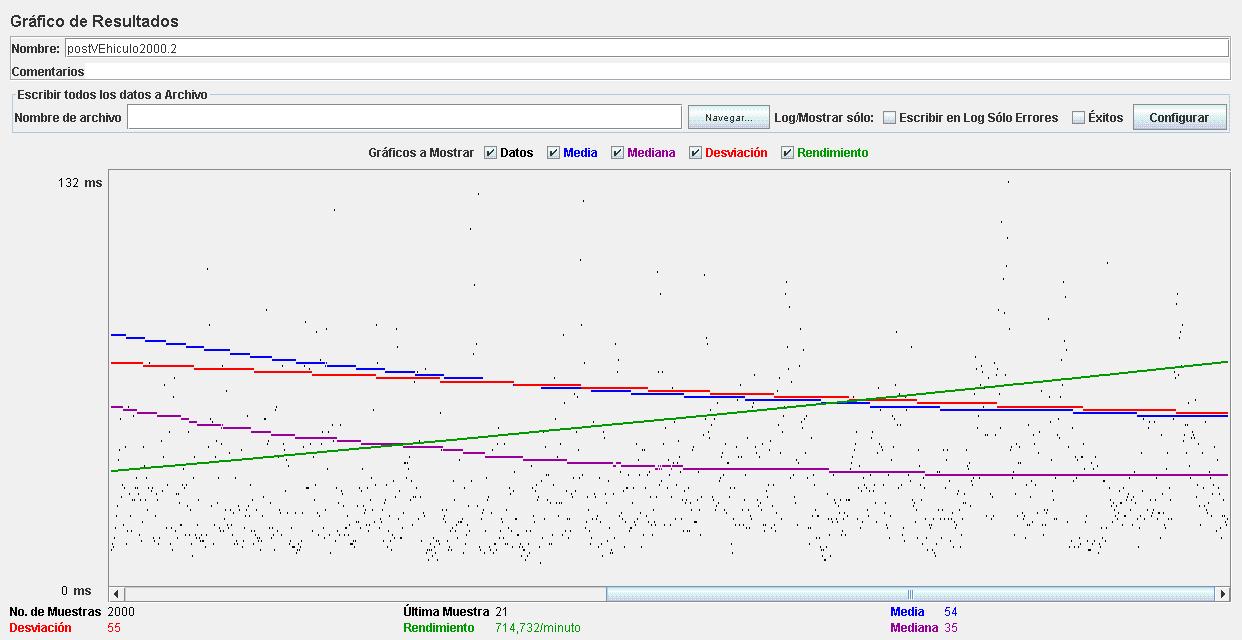


Post Vehiculo:

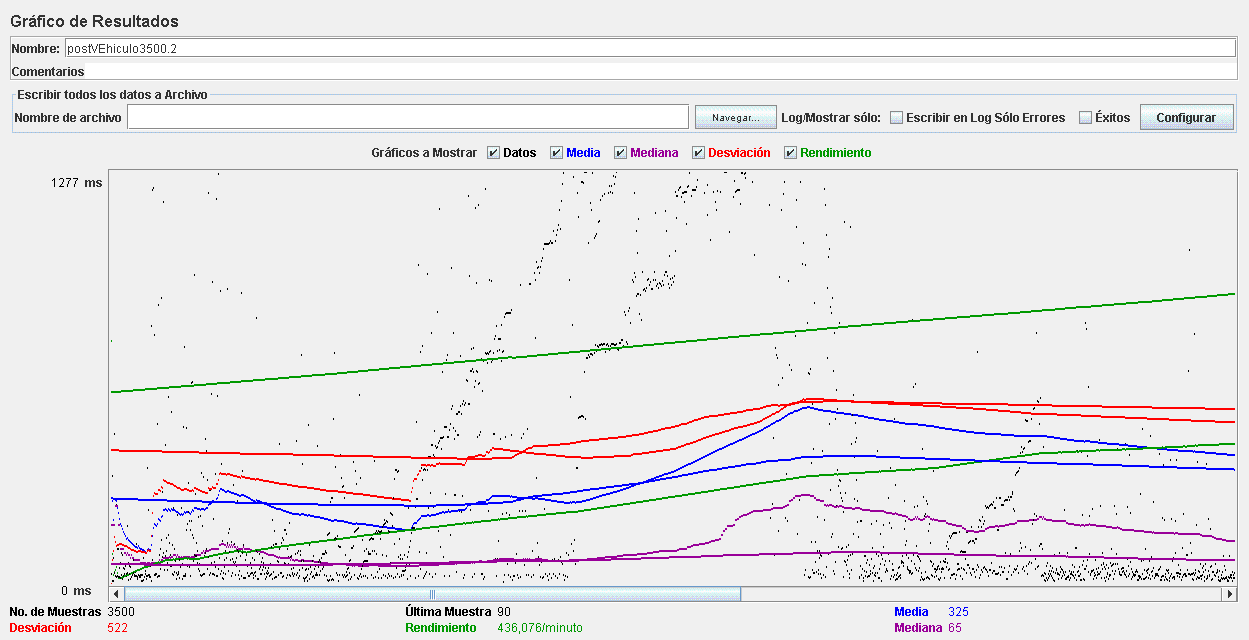
2000

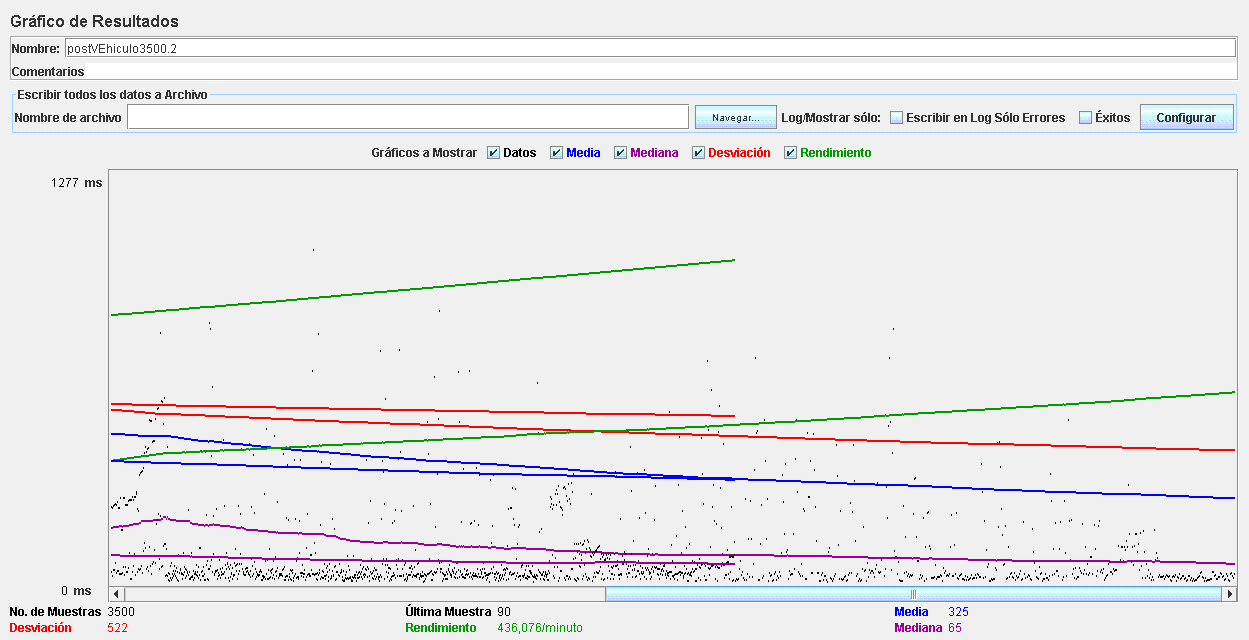
C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\postVehiculo2000.png





3500

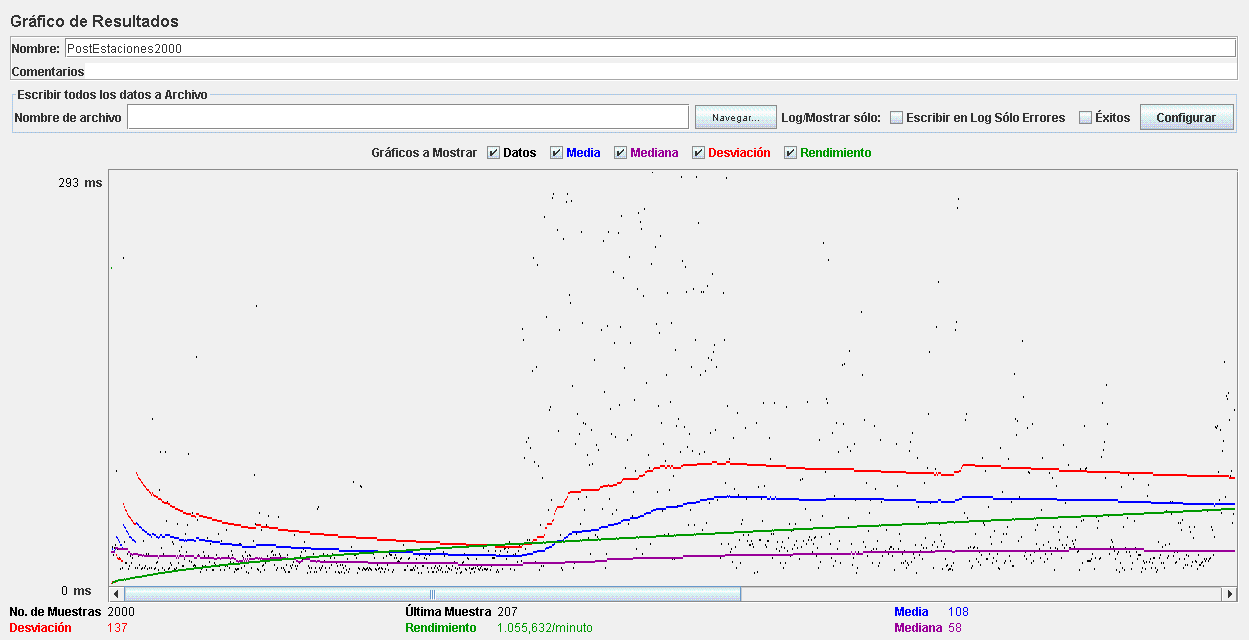


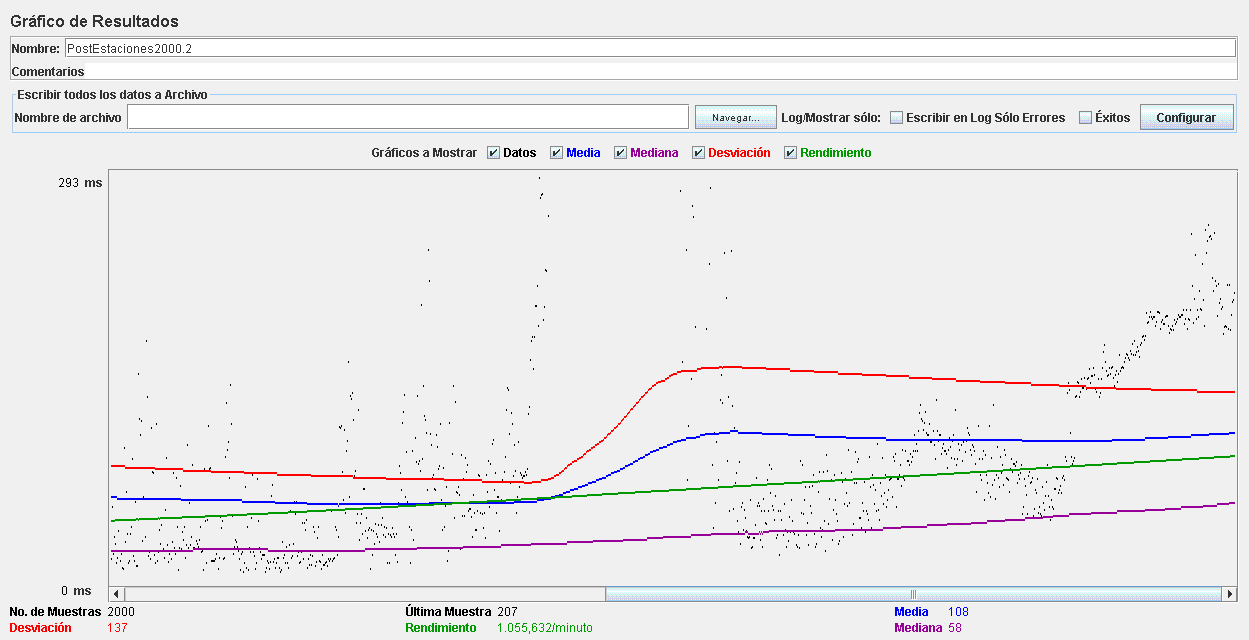


Post estaciones:

2000

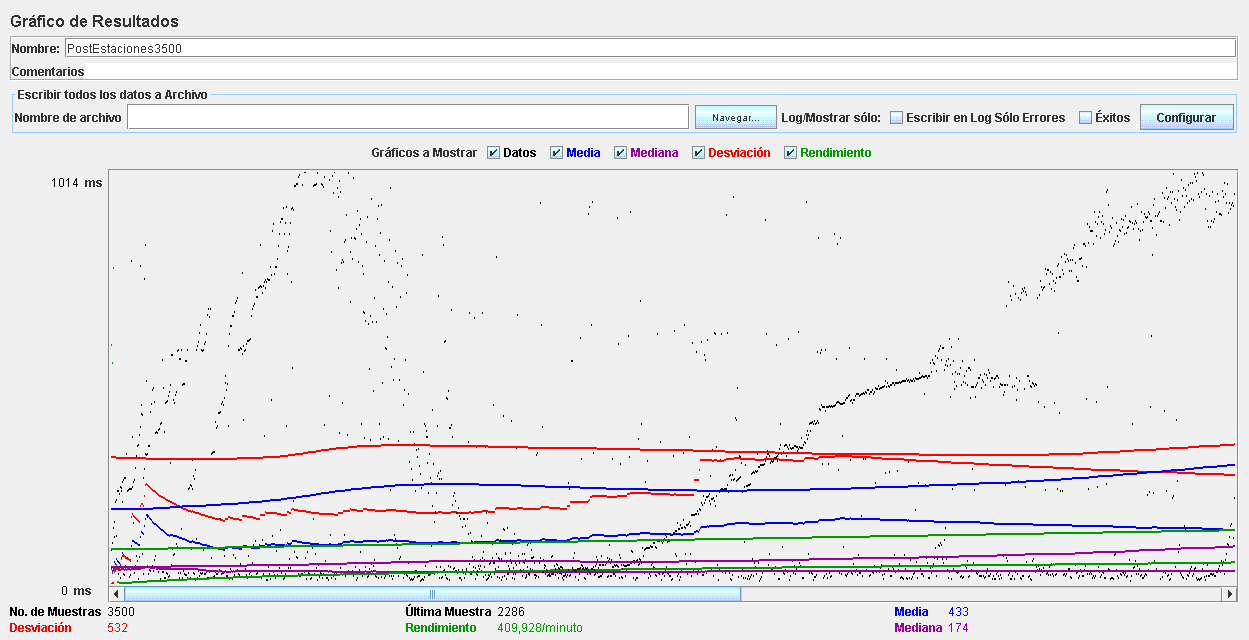
C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\posEstaciones2000.png

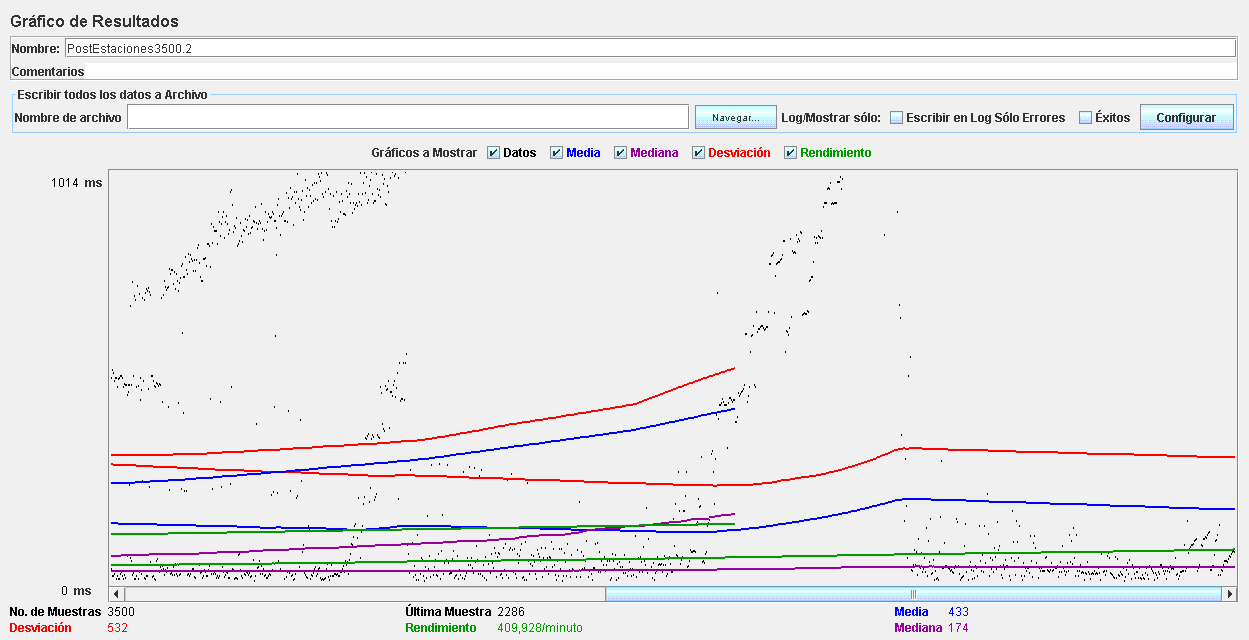




***3500***

C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\posEstaciones3500.png





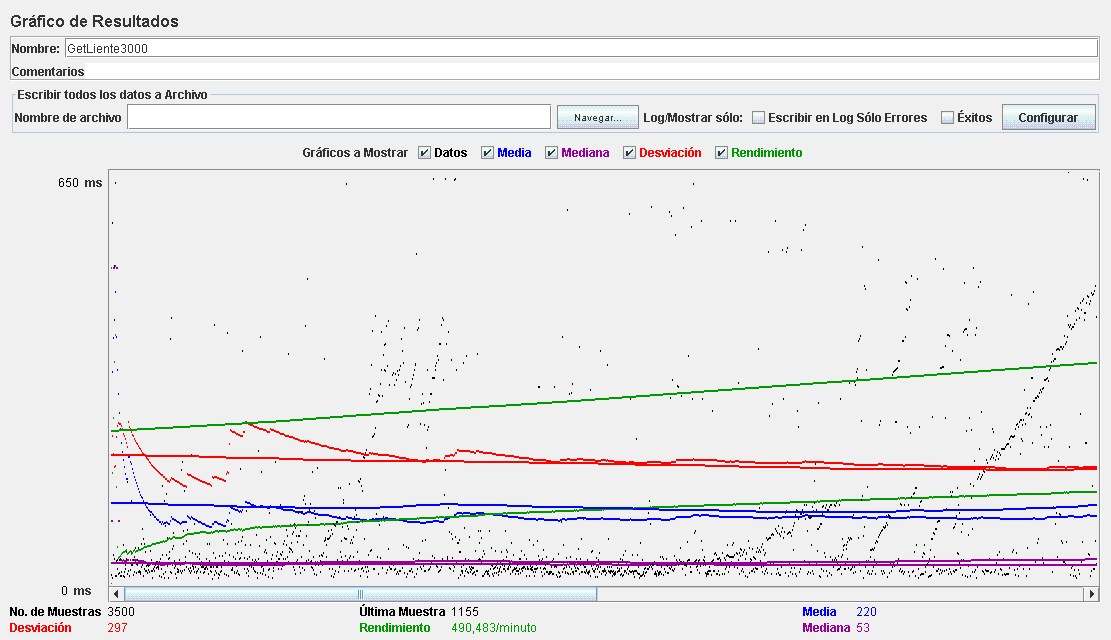
***GETS***

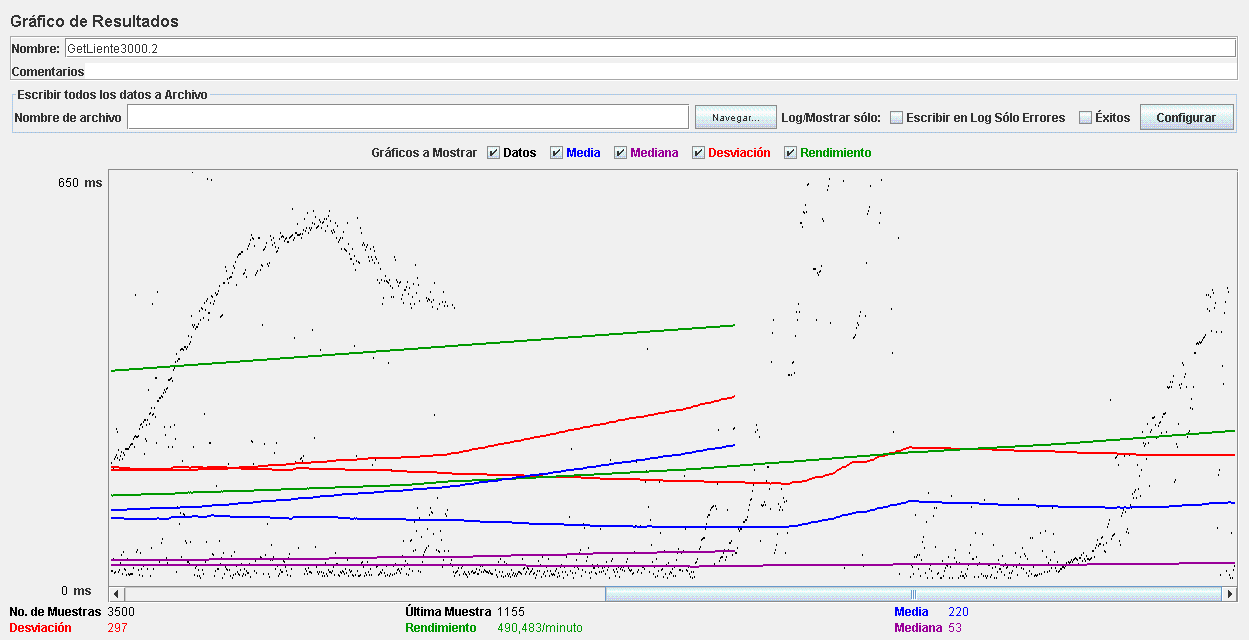
**DOSmaquinas**

GetCLientes

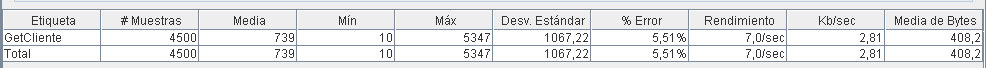
3500

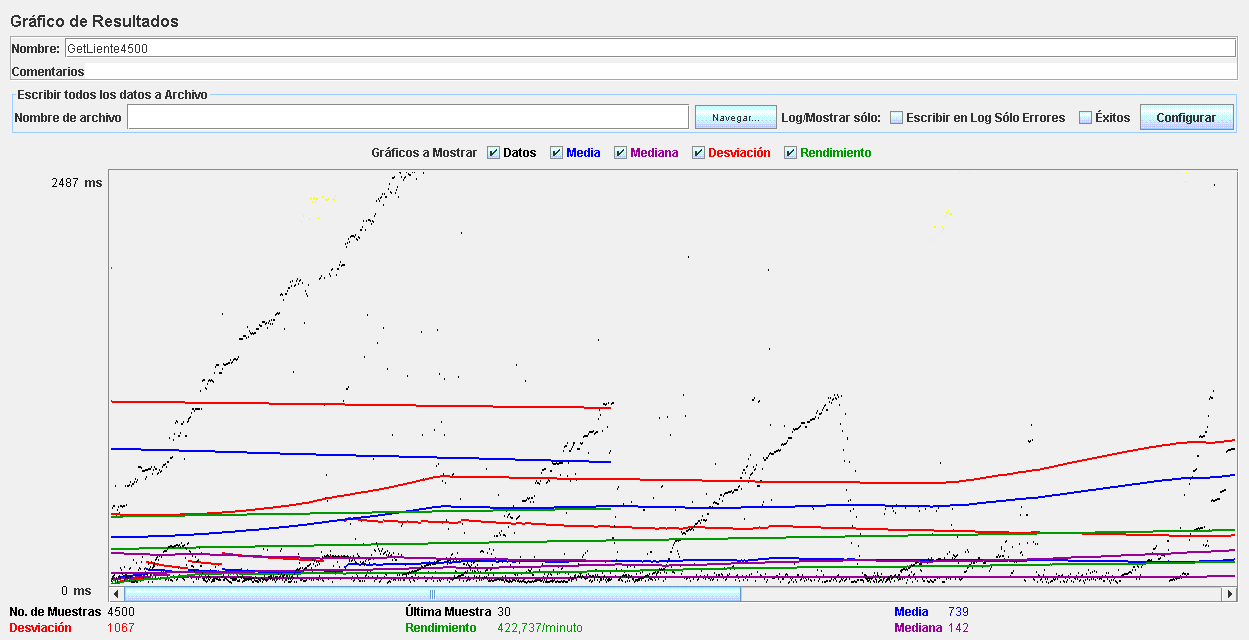
C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\getCLiente3000.png

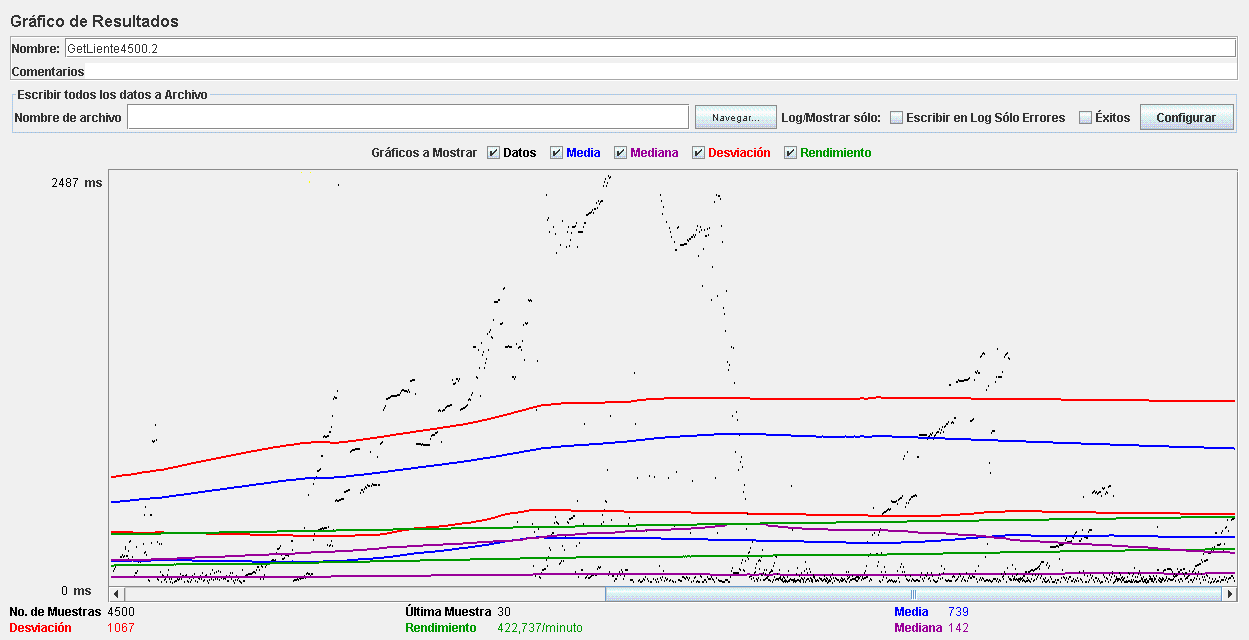




4500 ERROR

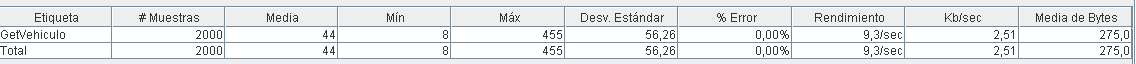


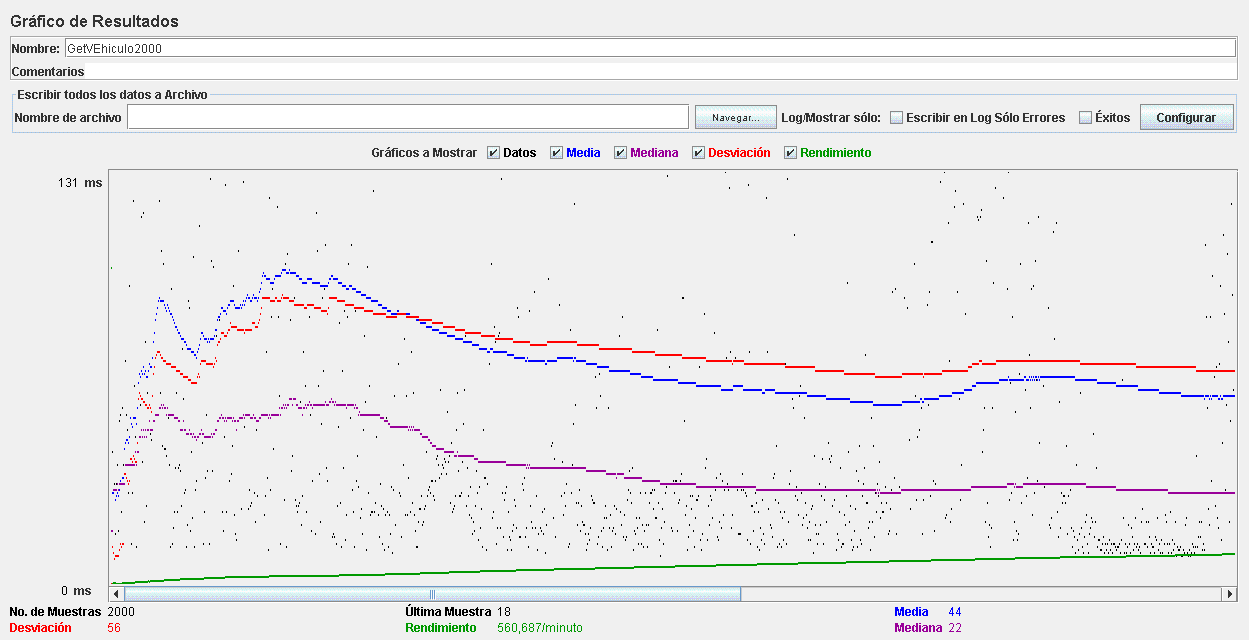


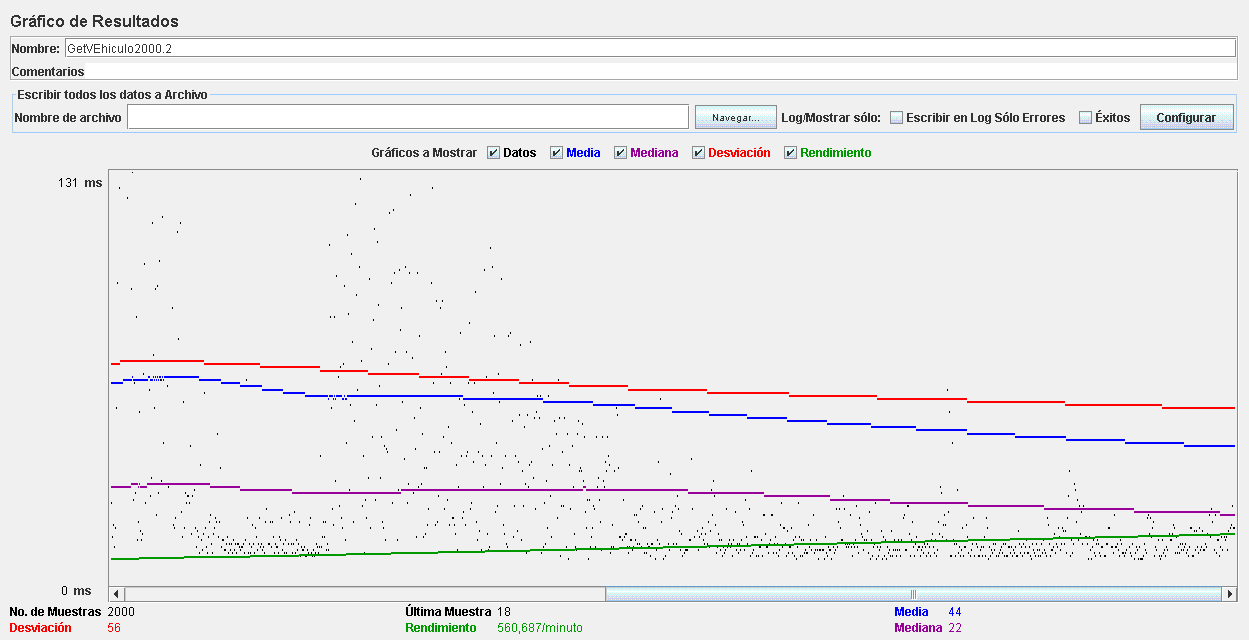


Get vehículos

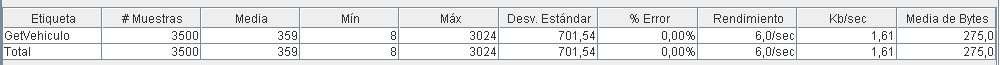
2000

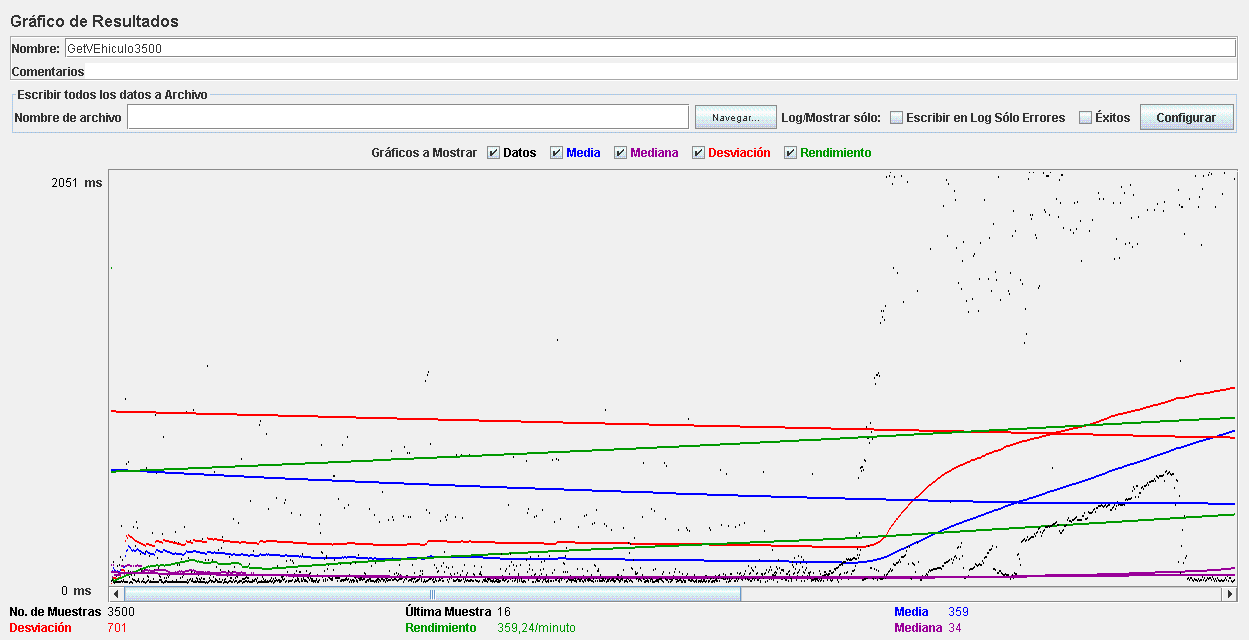


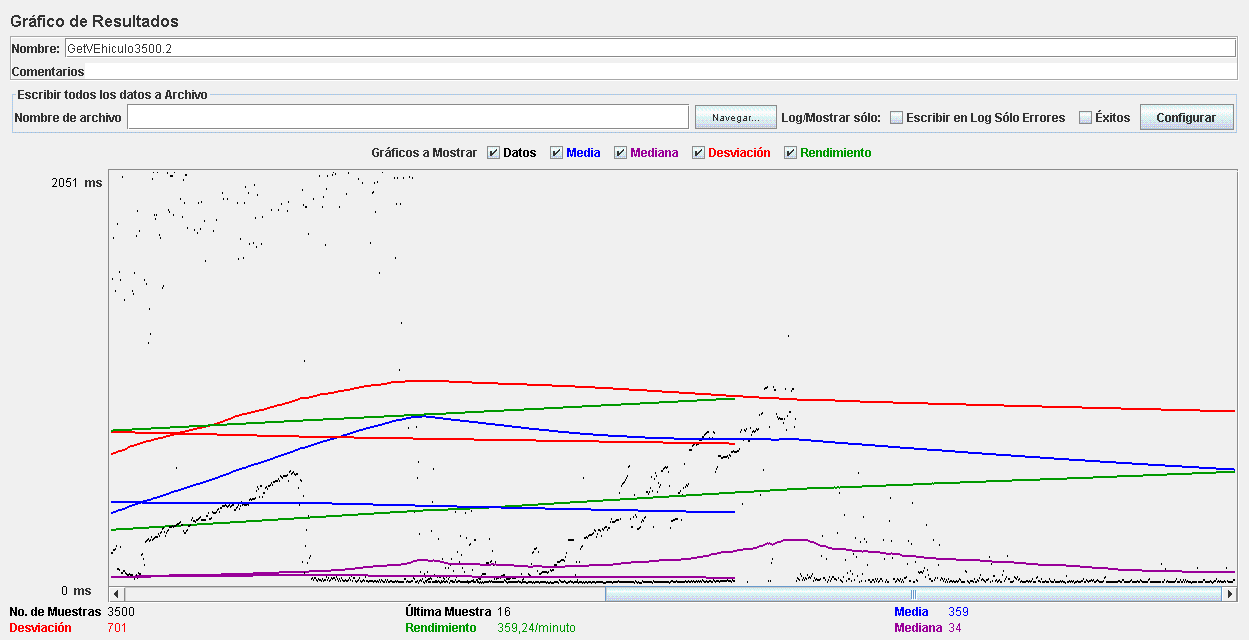




3500



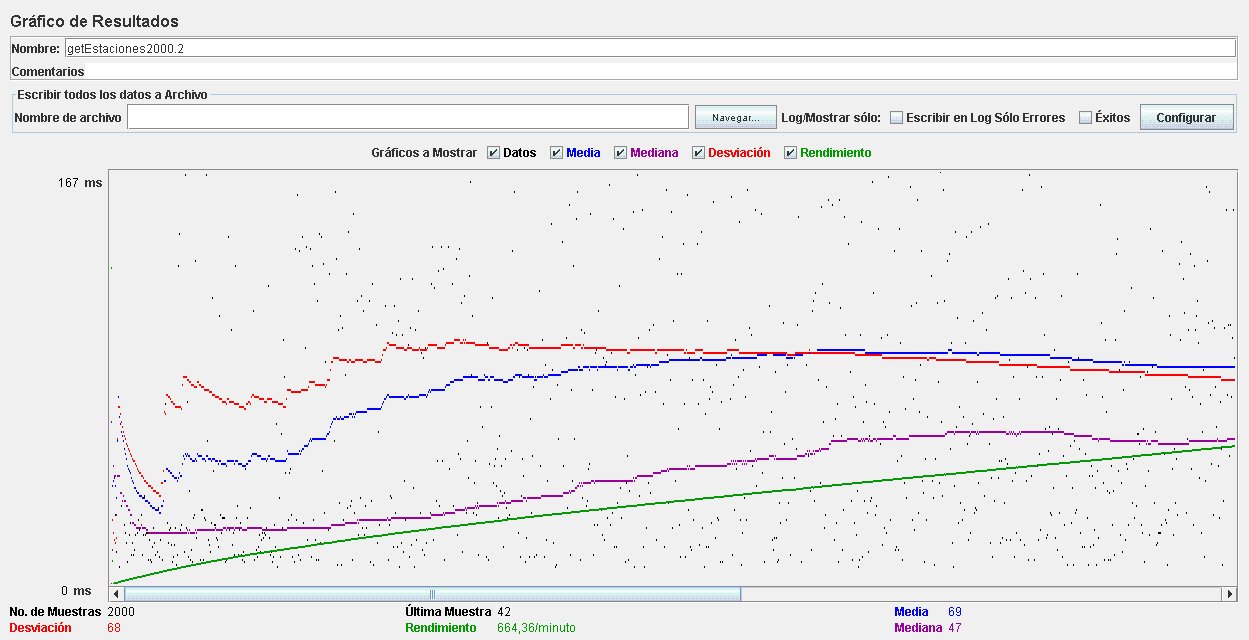


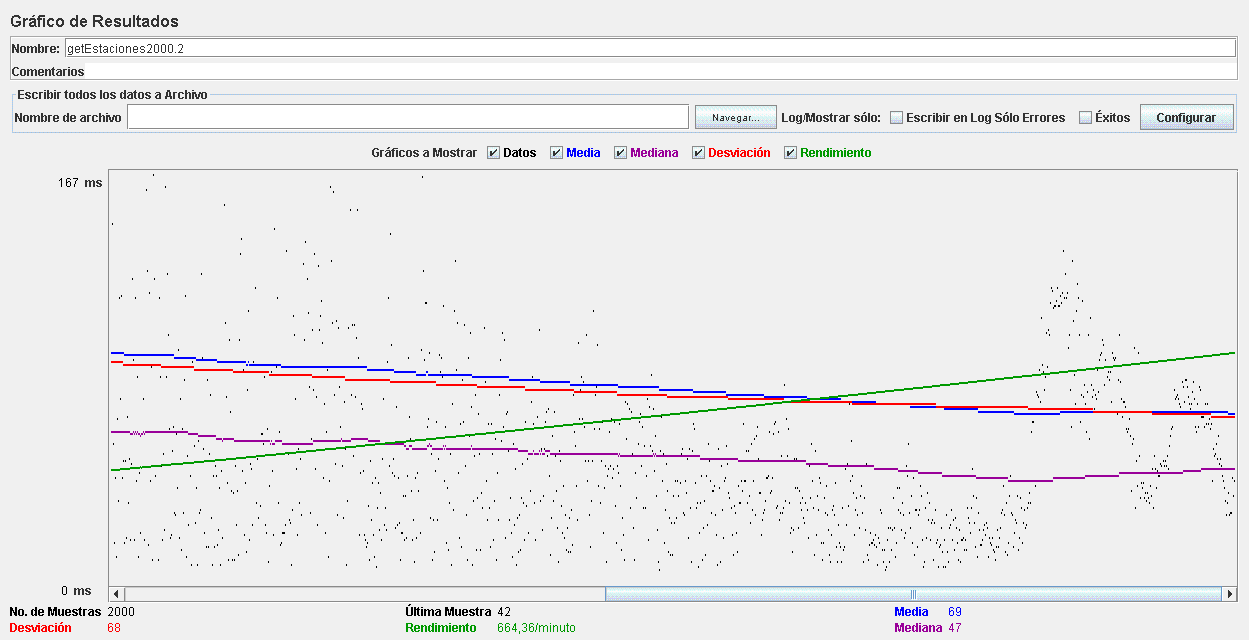


GET Estaciones

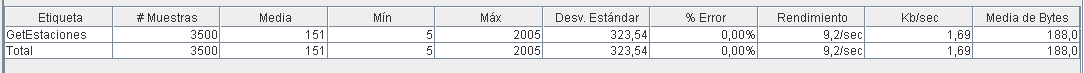
2000

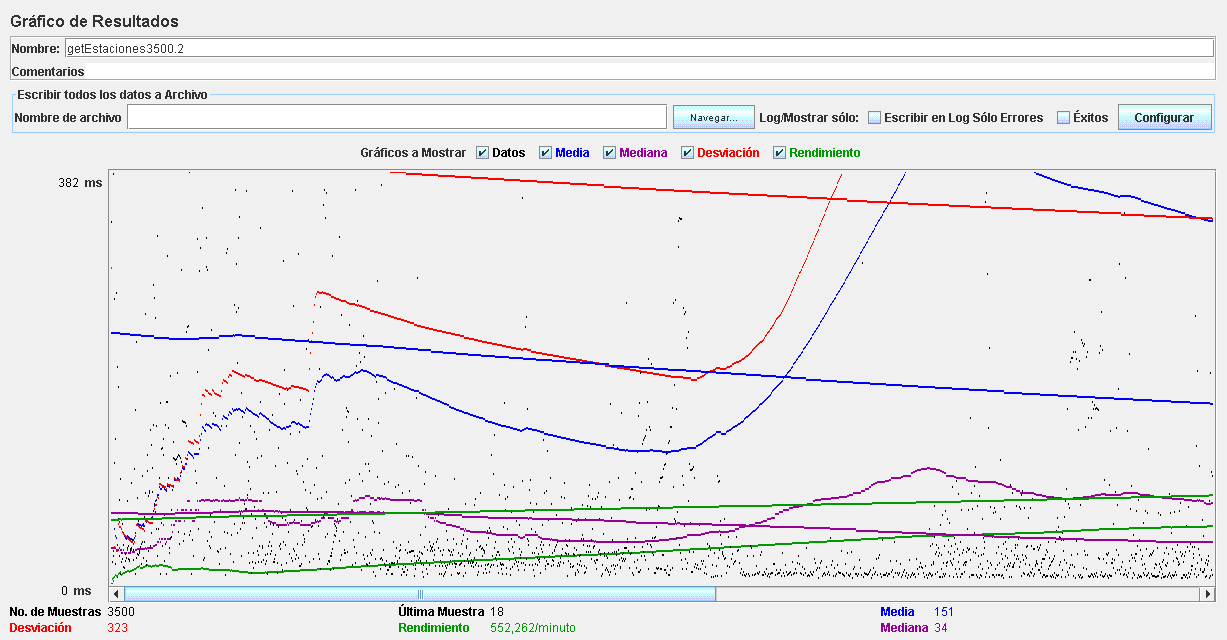
C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\getEstaciones2000.png

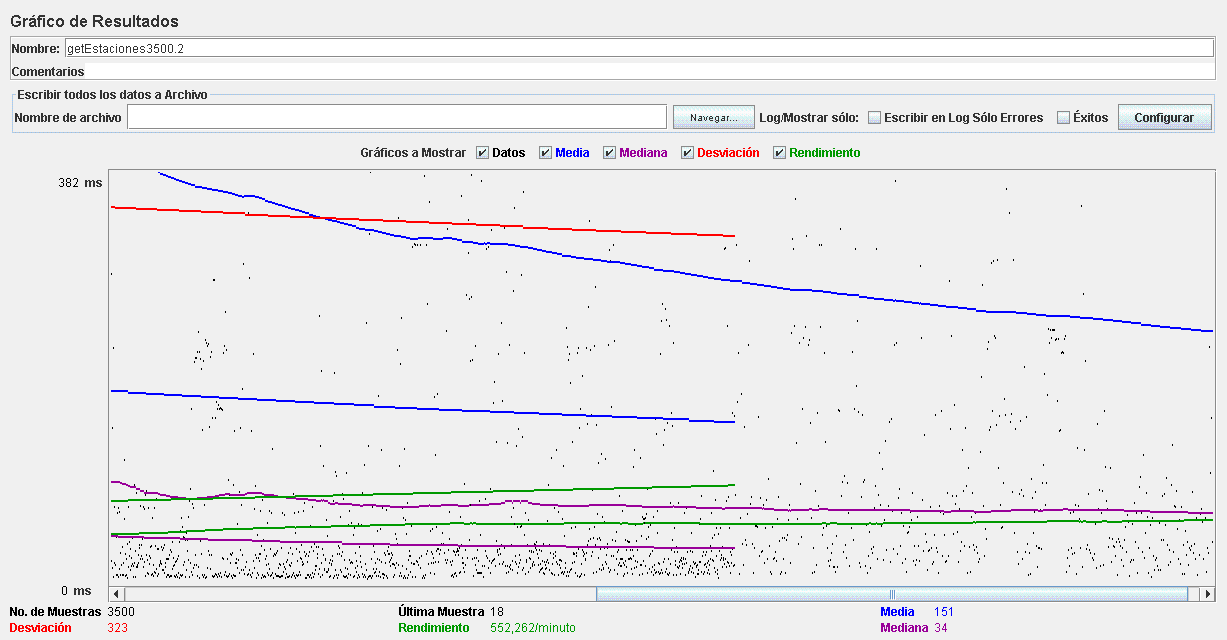




3500



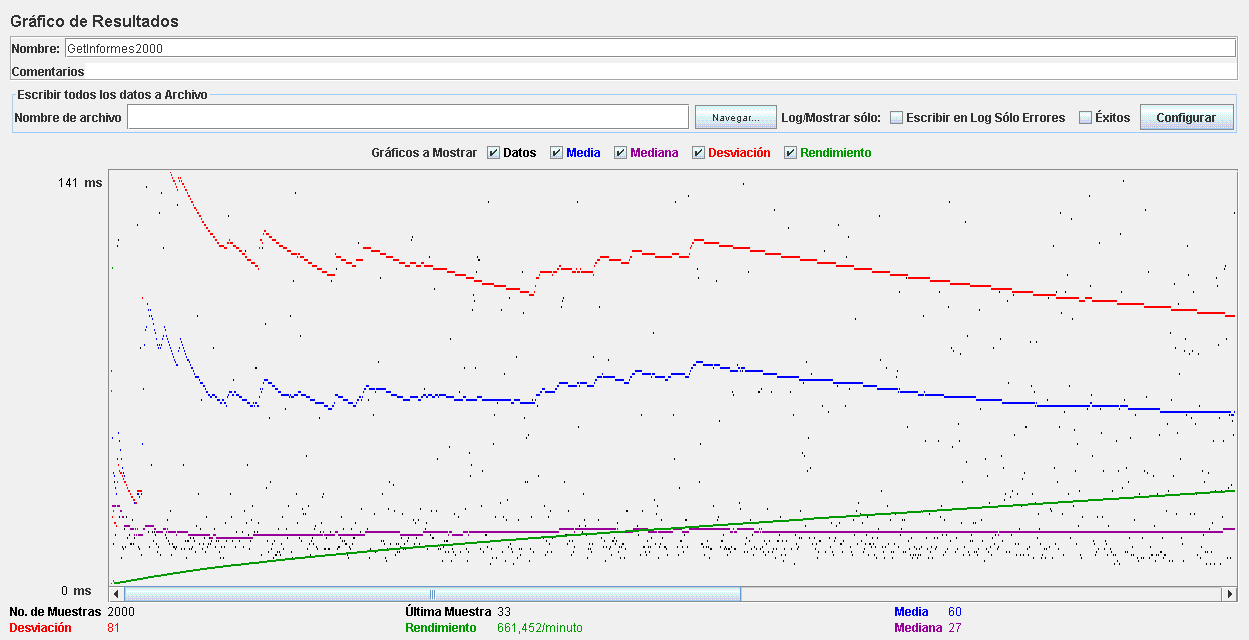


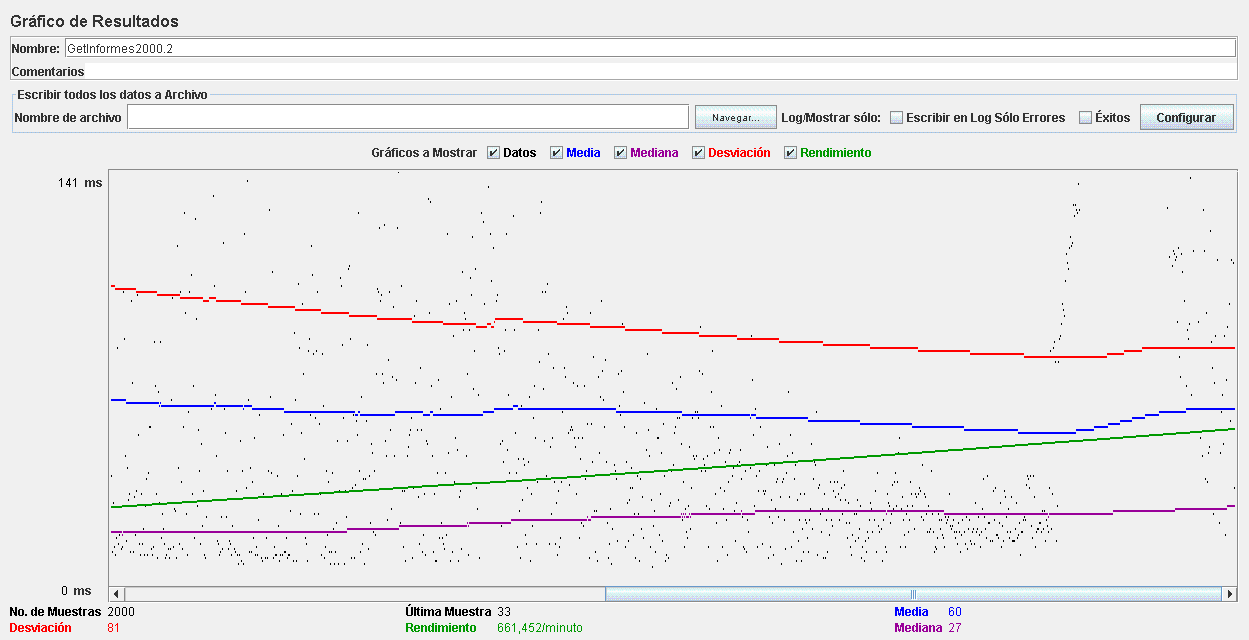


GET Informes

2000

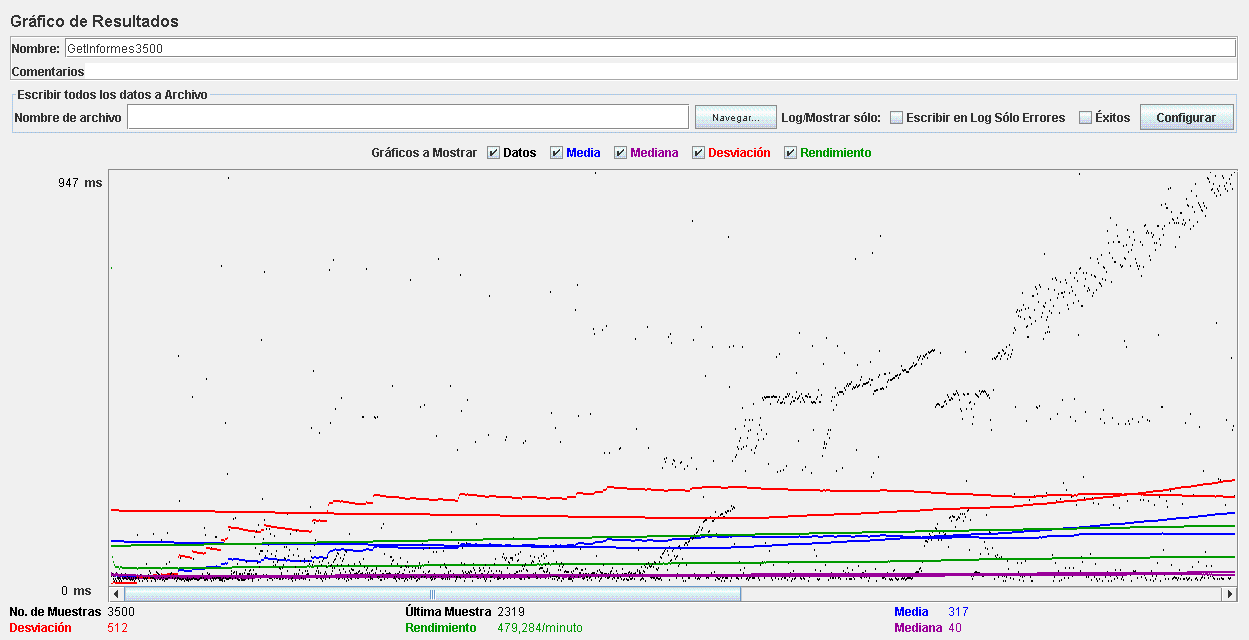
C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\getInformes2000.png

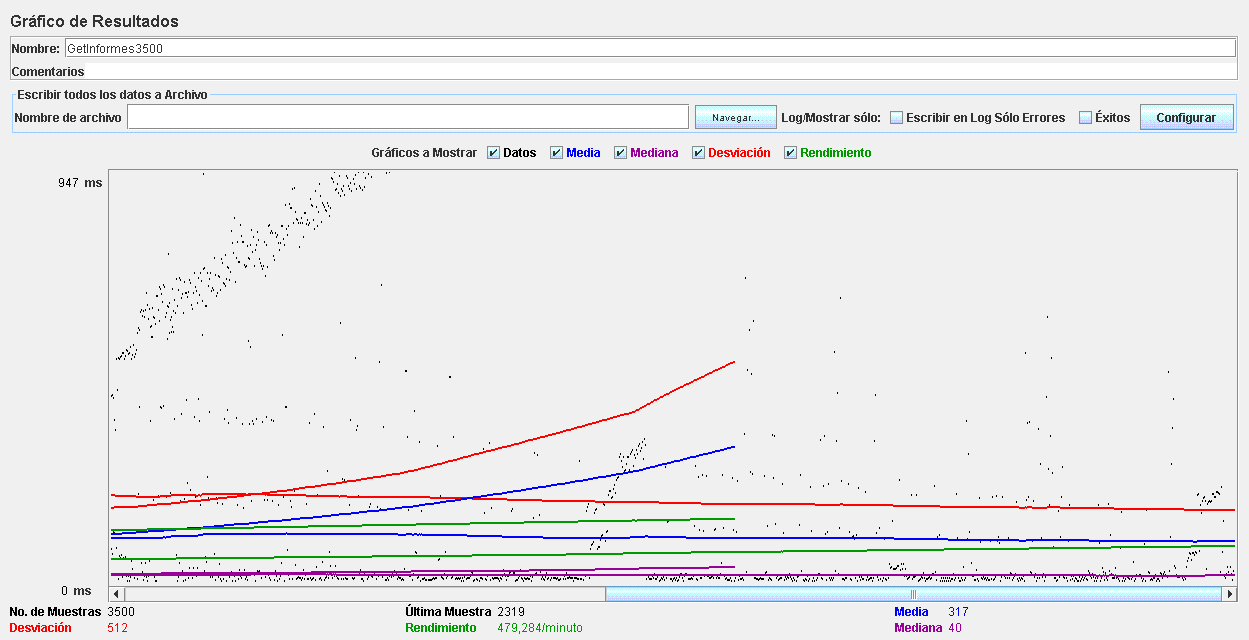




3500

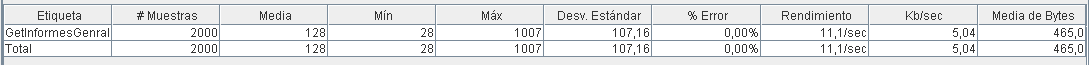
C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\getInformes3500.png

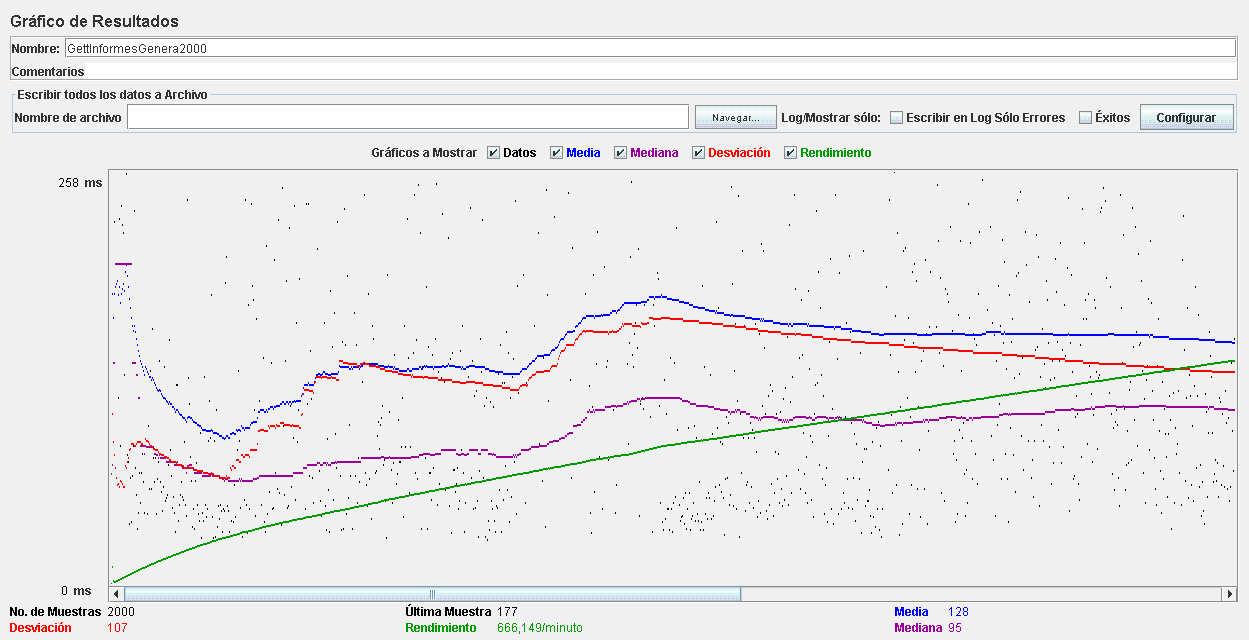


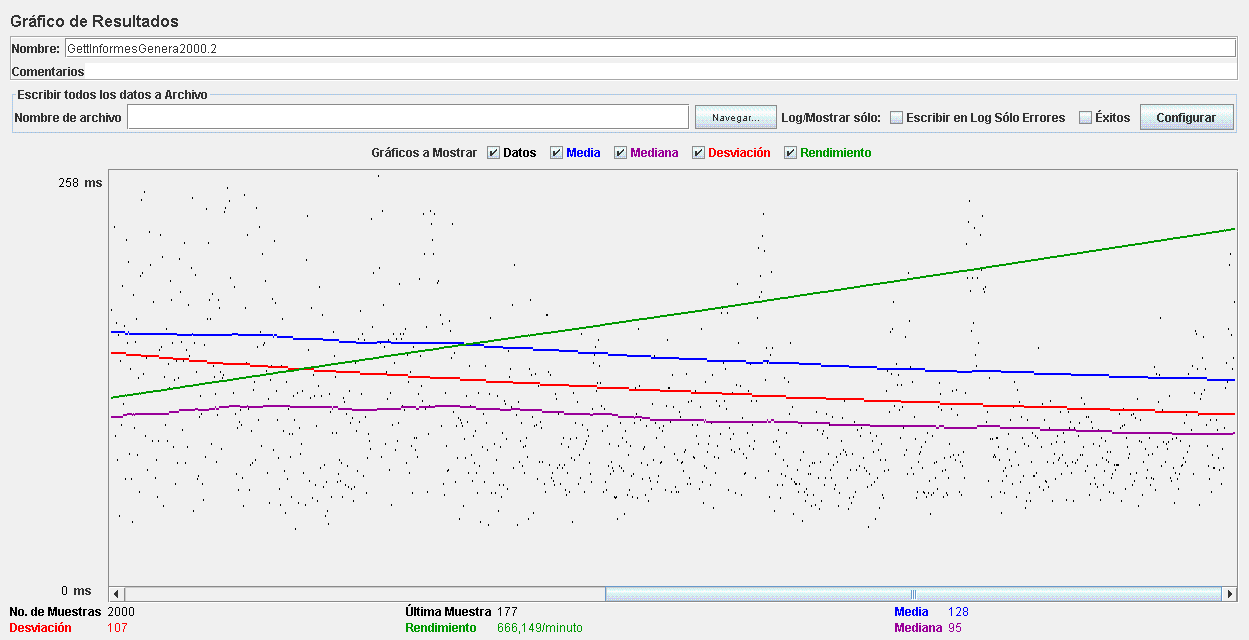


GET informe general

2000







3500

**Arquitectura Elegida 2.1**

Para esta entrega decidimos continuar usando play! con la configuración anteriormente encontrada puesto que su estilo arquitectural basado en actores y su modelo MVC completamente asincrónico y sin estado nos permite mantener tiempo reales de respuesta. Además, ya que mediante la configuración que vuelve la aplicación puramente asincrónica, logramos que la escalabilidad aumentara puesto que se podían responder un mayor número de peticiones en cierto rango de tiempo.

Por otra parte, nosotros decidimos seguir utilizando una base de datos relacional puesto que de esta manera lograríamos las propiedades ACID y por ende una mayor integridad de los datos que asegurarían unos buenos resultados en los servicios REST ofrecidos por nuestra aplicación. Además, cabe resaltar que esta decisión llevo a realizar un “trade-off” a favor del desempeño de la aplicación, y en contra de la escalabilidad de la misma puesto que dicho tipo de base de datos es muy rígido y estructurado.

Además, se decidió bajar la aplicación de la nube puesto que esta decisión estaba afectando mucho el desempeño de nuestra aplicación. Debido a lo anterior, pasamos a usar una base de datos desplegada localmente sobre PostgreSQL que aumento el desempeño de la apliciación pues la información ya no debe viajar hasta la nube. Nuevamente se decidió hacer un "trade-off" a favor del desempeño de la aplicación y en contra de la escalabilidad, conectividad y elasticidad de la misma debido a que la decisión de subir la aplicación a la nube bajo el desempeño demasiado, haciendo que dicha decisión no fuera buena.

Por último, se decidió utilizar Nginx como balanceador de carga de la aplicación con el fin de buscar una mayor disponibilidad en la misma. Es importante resaltar que la configuración round-robin reparte las peticiones equitativamente hacia todos los servidores, mientras que la configuración least-conn envía la petición al servidor con menor numero de conexiones activas. Debido a lo anterior se eligió una configuración least-conn pues esta ayuda más en el desempeño, escalabilidad y disponibilidad de la misma debido a su manera "inteligente" de repartir las peticiones.

**Comparación con entrega pasada 1.2 - 2.1**

Para esta entrega hubo varios cambios importantes. En primer lugar, se bajo la aplicación de la nube con el fin de mejorar el desempeño de la misma debido a lo explicado en el punto anterior. Además, debido a lo anterior, se tuvo que desplegar la capa de persistencia localmente en PostgreSQL lo cual nuevamente ayudo con el desempeño de nuestra aplicación. No obstante, cabe resaltar que todo lo anterior mejoro en un contexto que no implica la internacionalización de nuestra aplicación porque de ser así la nube brindaría muchas más ventajas debido a su buen manejo de escalabilidad, conectividad y elasticidad.

Por otra parte, se comenzó a utilizar un balanceador de carga con configuración least-conn que ayudo a recuperar un poco la escalabilidad perdida al bajar la aplicación de la nube, y que además agrego disponibilidad a la misma. No obstante, esta decisión afecto un poco el desempeño de la aplicación debido a que la información debe ser re direccionada entre los diversos servidores, y en caso de que alguno no se encuentre funcionando se demora un poco el balanceador de carga en establecer que debe mandárselo a otro servidor. No obstante, cuando están todos los servidores funcionando el desempeño aumenta puesto que se pueden procesar las peticiones más rápidamente entre varios servidores. Se decidió que al haber bajado la aplicación de la nube y haber recuperado bastante desempeño, era justo y necesario realizar este "trade-off" puesto que ahora la aplicación necesitaba mostrar más disponibilidad incluso si en algunos casos se sacrificaba desempeño.

También, se termino de conectar el front-end con el back-end de la aplicación y se dejaron todos los servicios expuestos por la misma funcionales a través de esta capa de presentación. Por último, tras realizar las pruebas pudo verse la mejora en el desempeño, escalabilidad y disponibilidad de la aplicación. Lo anterior se puede entender por las decisiones tomadas a lo largo de esta entrega. No obstante, las pruebas siguieron presentando problemas en los servicios PUT & DELETE debido a que las pruebas realizadas por JMeter intentan actualizar/borrar el mismo objeto tantas veces como peticiones se le ponga, lo que nos genera una gran problemática puesto que al intentar todos interactuar con un mismo objeto se vuelve un servicio plenamente sincrónico que genera una cola en la base de datos y por ende unos resultados malos.

**Conclusiones 2.1**

Se llega a la conclusión de que importante volver a analizar decisiones arquitecturales tomadas anteriormente puesto que con la entrada de nuevos requerimientos no funcionales puede que toque re-definir, dejar o quitar dichas decisiones. Además, se concluyo que es de gran importancia tener muy en cuenta los famosos "trade-off" debido a que realizar una decisión que afecte un atributo de calidad como la disponibilidad pero que mejore otro como el desempeño, puede llevar a que toque realizar otro "trade-off" en otra parte de la

arquitectura que ayude en este caso a la disponibilidad y afecte un poco el desempeño, siempre buscando mantener un balance entre los atributos de calidad requeridos. Por ejemplo como ocurrió en nuestro caso al bajar la aplicación de la nube, llevo a que se comenzara a utilizar un balanceador de carga como lo es Nginx. Todo lo anterior llevo a resaltar la conclusión del experimento 1, es decir la gran importancia de un arquitecto de software en un proyecto puesto que sus decisiones pueden ser vitales para determinar si un proyecto se mantendrá a flote o no.

# References

N.A. (S.f). *Using nginx as HTTP load balancer*. Obtenido de NGINX: http://nginx.org/en/docs/http/load\_balancing.html