POLITECNICO
MILANO 1863

M.Sc. Computer Science and Engineering

Computer Systems Performance Evaluation

Progetto A

Set di parametri: 4

Leonardo Guerra

Codice persona: 10524955

Indice

Capitolo 1

Problema

1.1 Testo (in inglese)

A distributed architectural rendering algorithm creates images that show a view of a 3D reconstruction of a building.

It works in the following way. There are N processes, each one renders a view of the considered project. After a start-up time of exponentially distributed duration Z, each process first reads the data for the rendering from the storage. Then it splits each image into M smaller rectangular portions that are rendered separately on different threads. Rendering is performed by K identical servers. When all the M portions have been completed, the image is saved back on the storage. Both the servers and the storage can be considered processor sharing systems with exponential service time, and the start-up time can be modeled with a delay station. The average time required to read the data for a rendering is D_L, the time for rendering one of the M portions on one of the K servers is D_R, and the time required to save the final image on the storage is D_S.

The administrators wants to decide the minimum number of servers K_{min} and the best number of processes N to have an average rendering time less than R_{max}.

1.2 Parametri

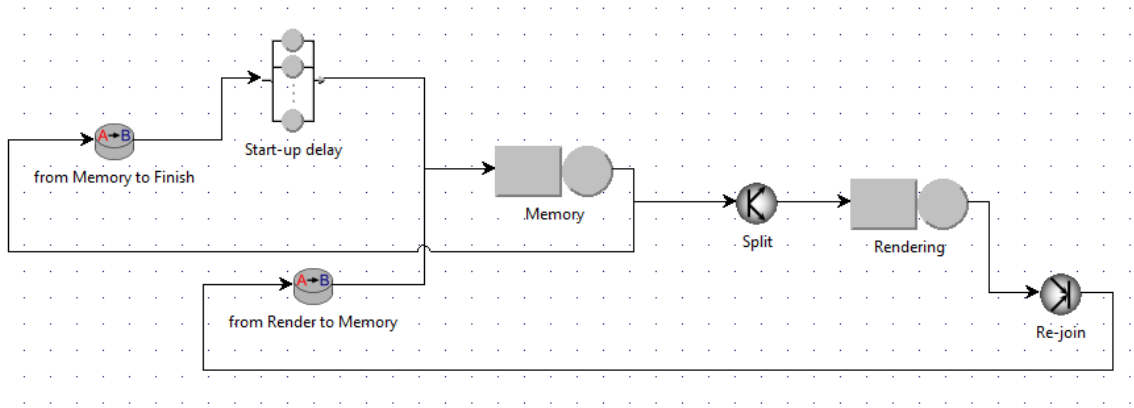
I parametri utilizzati sono stati quelli relativi al dataset 4 (tutti i millisecondi sono stati convertiti in secondi):

- $Z = 2$ s;
- $M = 192$;
- $D_L = 1.2$ s;
- $D_R = 0.2$ s;
- $D_S = 1.5$ s;
- $R_{max} = 20$ s.

Capitolo 2

Modello

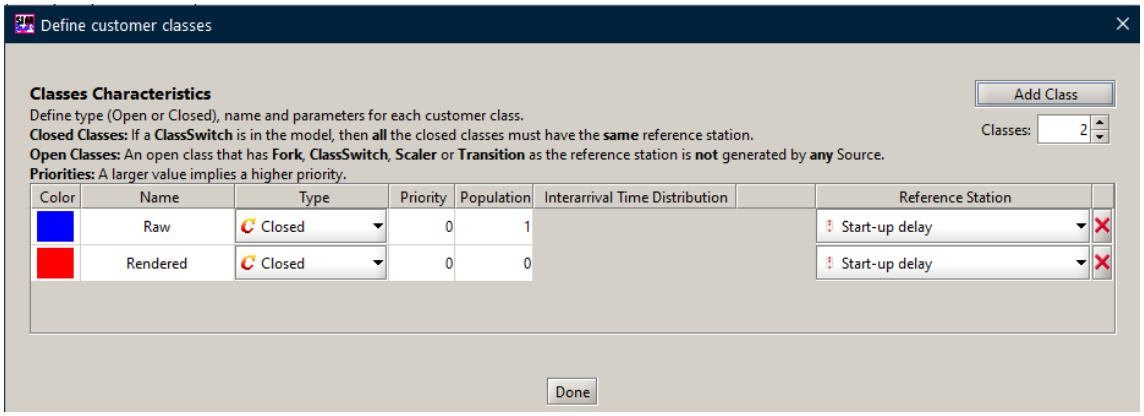
2.1 Rappresentazione grafica



2.2 Introduzione e classi

Il sistema è stato modellato con un multiclass closed queuing network.
Le classi del sistema sono due:

- **Raw**: che serve a modellare i job prima del rendering;
- **Rendered**: che modella i job dopo il rendering pronti per essere salvati in memoria.



Classi del sistema: Raw e Rendered

2.3 Stazioni

Il modello ha diverse classi (con nomi, per quanto possibile, già autoesplicativi):

- **Start-up delay**: è la reference station, ed è una delay station che modella il tempo di start-up all'avvio, di durata Z = 2 secondi, distribuito esponenzialmente; tutti i job sono instradati alla stazione Memory;
- **Memory**: è una queue station (con coda infinita) con un solo server; il service time per la classe Raw è distribuito esponenzialmente con media D_L = 1.2 secondi, e, sempre distribuito esponenzialmente ma con media D_S = 1.5 secondi, per la classe Rendered; i job della classe Raw sono instradati (tutti e soli) verso la stazione di fork Split, mentre i job della classe Rendered sono instradati verso il class switch che precede la stazione Start-up delay;
- **from Memory to Finish**: è un class switch che trasforma i job da classe Rendered a classe Raw, per sancire il termine dell'intero processo di rendering del job in questione;
- **Split**: è una stazione di fork che divide ogni job che riceve in M = 192 task;
- **Rendering**: è una queue station (con coda infinita) con k server; il service time per i job della classe Raw è distribuito esponenzialmente con media D_R = 0.2 secondi, mentre i job di classe Rendered non arrivano mai in questa stazione; tutti i task vengono instradati alla stazione di join Re-join;
- **Re-join**: è una stazione di join che raccoglie tutti i task di uno stesso job per terminarne la fase di rendering;
- **from Render to Memory**: è un class switch che trasforma i job da classe Raw a classe Rendered, per indicare che il job deve essere salvato in memoria dopo essere stato processato in Rendering.

Capitolo 3

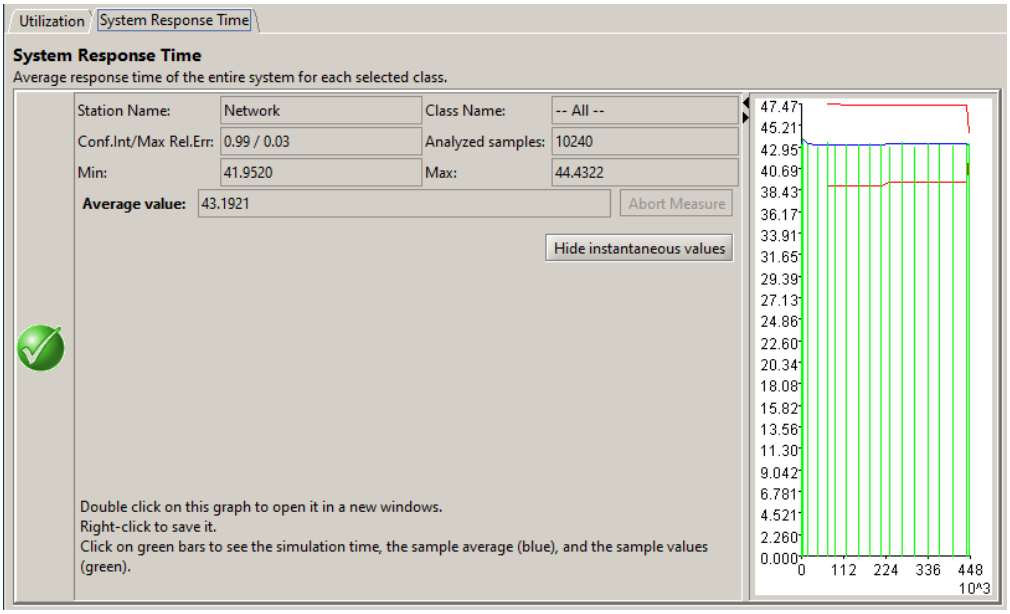
Analisi del modello

3.1 Introduzione

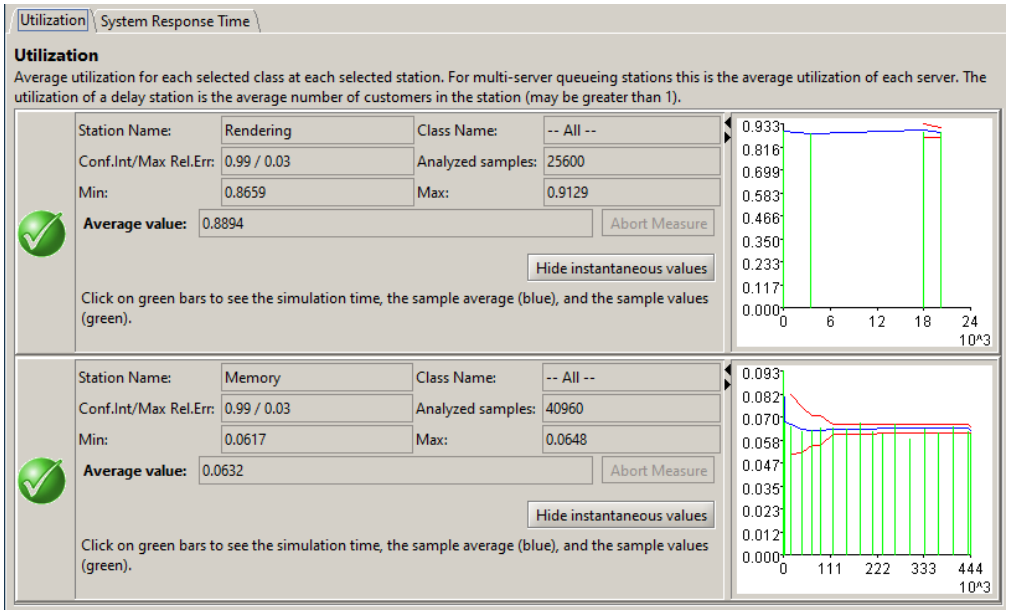
Il modello è stato analizzato con il simulatore JSIMgraph (della suite Java Modeling Tools, JMT), non essendo possibile utilizzare le tecniche analitiche (Mean Value Analysis), a causa della presenza di server multipli (che invalida la Service Time homogeneity) e dell'utilizzo della Fork e della Join.
Nelle queue è stata usata la queuing policy First Come First Served (FCFS).

3.2 Analisi di partenza

Come punto di partenza dell'analisi sono partito con un solo processo (N=1) e la stazione Rendering con server singolo, ottenendo un risultato "negativo" in termini di Response time del sistema, il cui valore medio (circa 43 secondi) era ben oltre l'obiettivo prefissato di rimanere al di sotto dei 20 secondi. Il sistema era comunque stabile, essendo la Utilization della stazione Rendering pari circa a 0.89, quindi minore di 1:



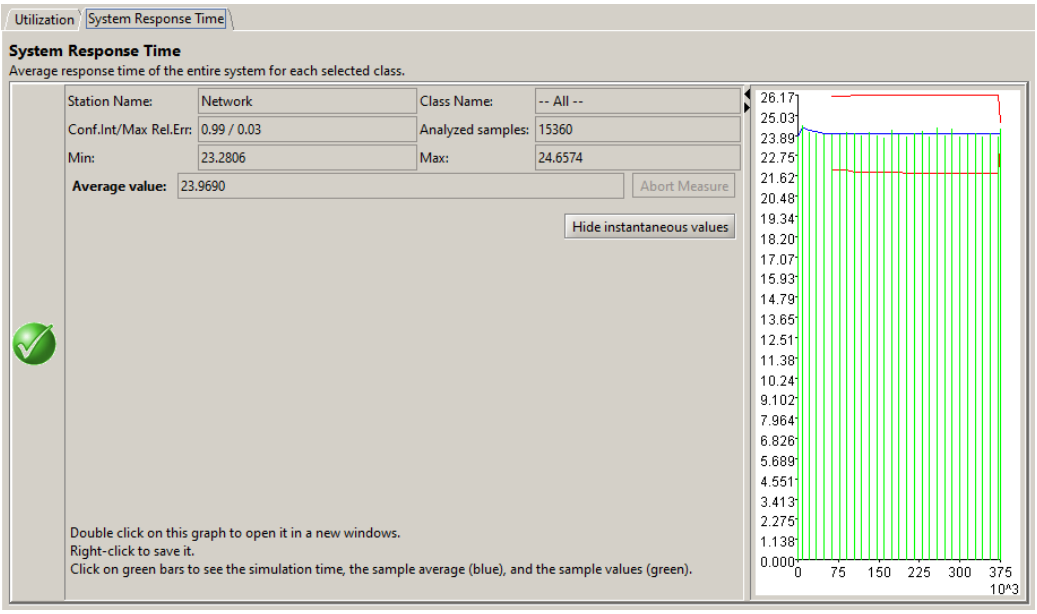
Response time del sistema con k=1 e N=1



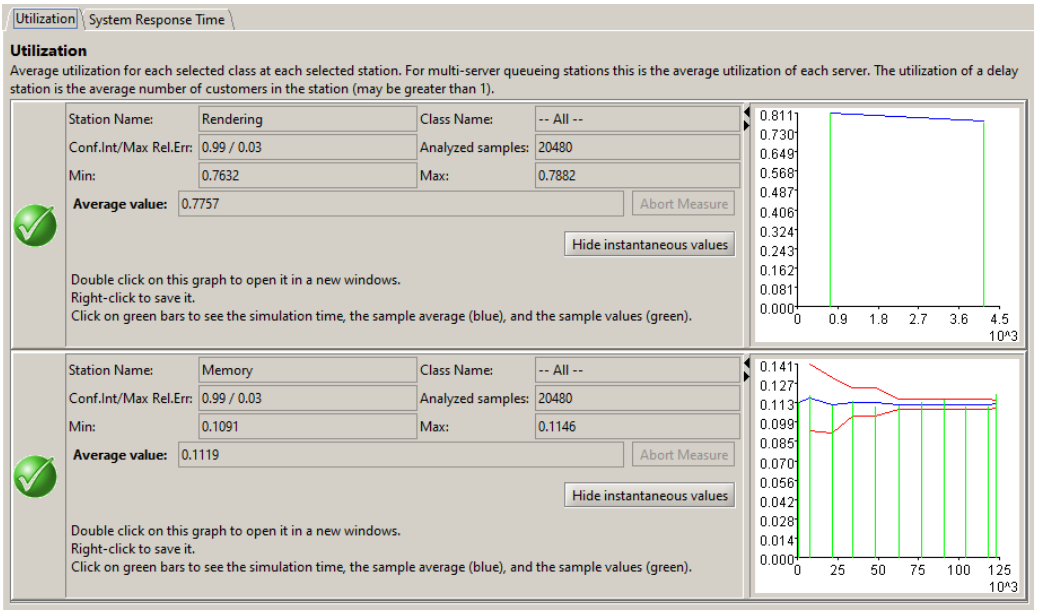
Utilization di Rendering con k=1 e N=1

Nelle immagini è anche riportata per completezza la Utilization della stazione Memory, nonostante questa sia sempre molto bassa, ben lontana da 1.

Quindi, ho continuato l'analisi provando ad aumentare a 2 il numero di server per Rendering (ancora con N=1), ottenendo un Response time del sistema (pari a circa 24 secondi) poco superiore al valore prefissato, ma sempre mantenendo il sistema stabile (Utilization di Rendering pari circa a 0.76, minore di 1):

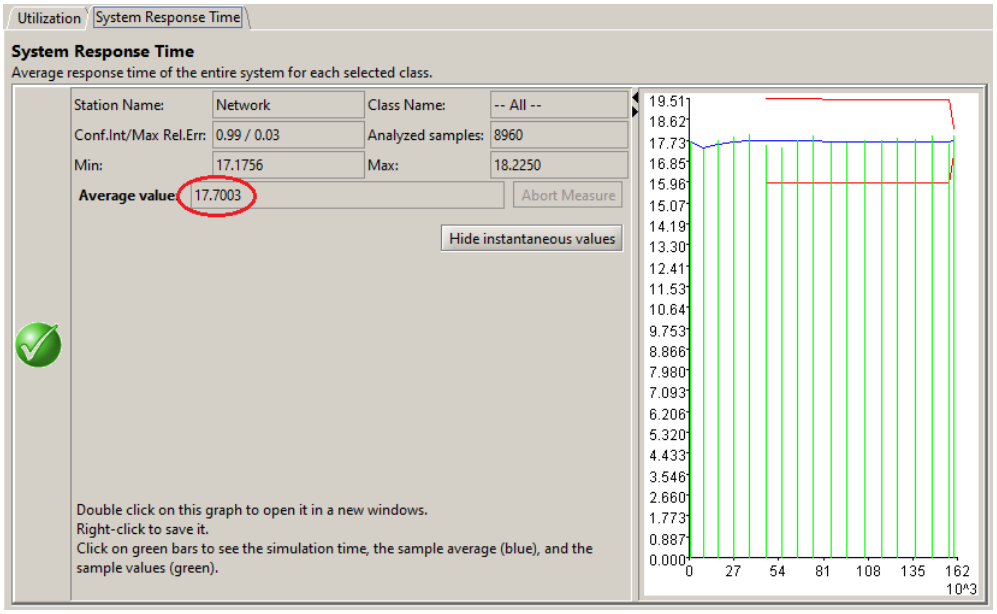


Response time del sistema con k=2 e N=1

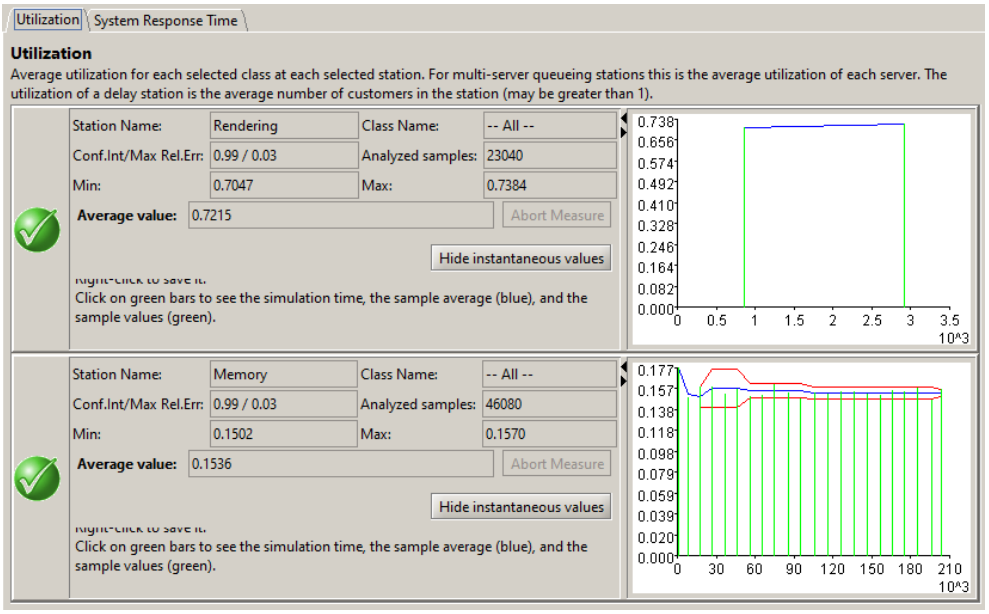


Utilization di Rendering con k=2 e N=1

Infine, provando con 3 server in Rendering, sono finalmente riuscito a centrare l'obiettivo di mantenere il Response time del sistema al di sotto dei 20 secondi (ottenendo un valore di circa 17.7 secondi), sempre con una Utilization di Rendering minore di 1 (pari a circa 0.72):

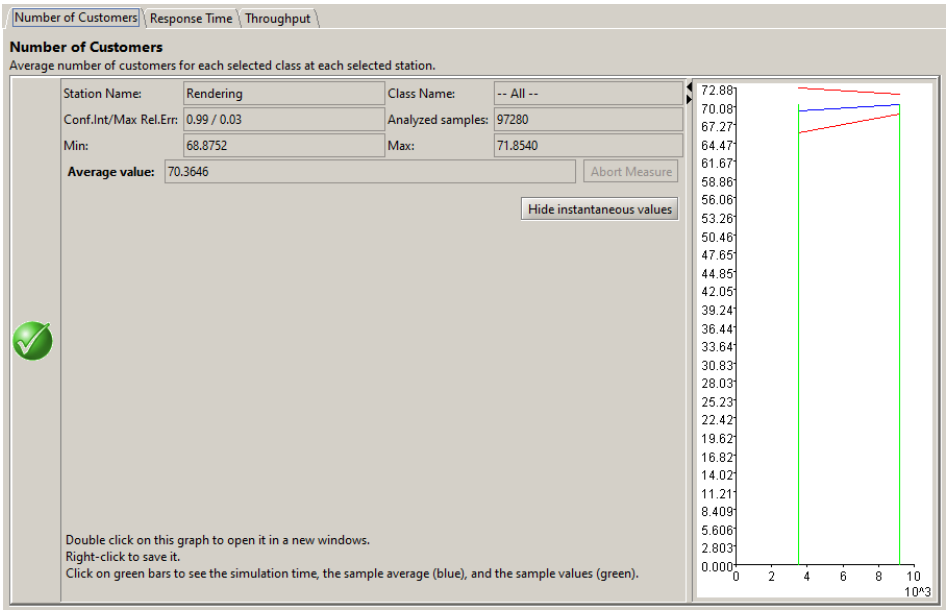


Response time del sistema con k=3 e N=1

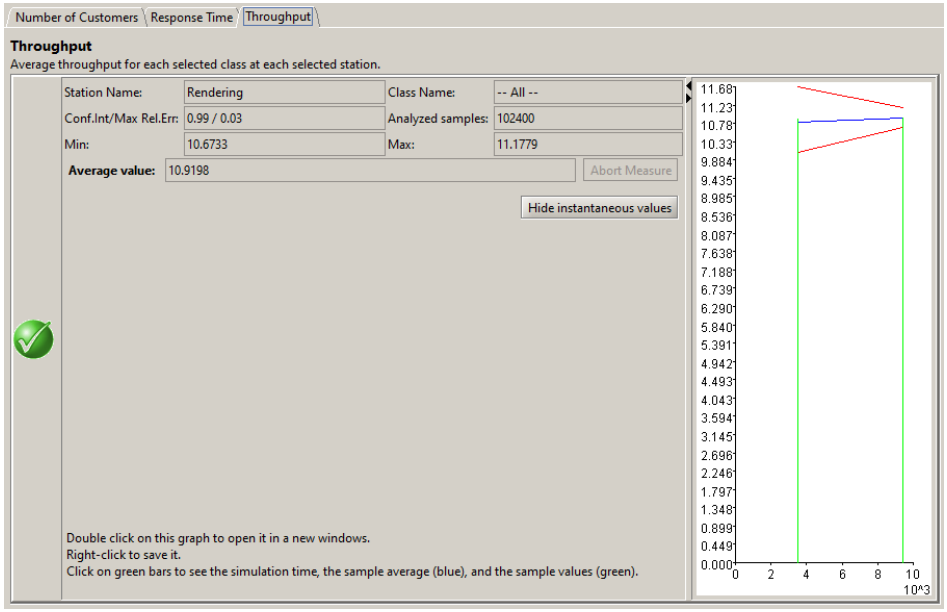


Utilization di Rendering con k=3 e N=1

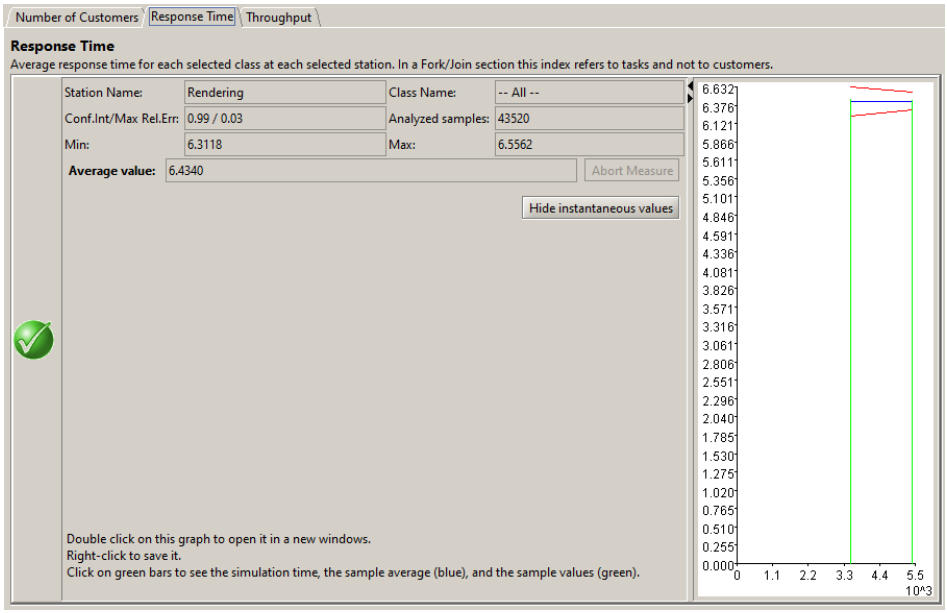
Per il caso con 3 server in Rendering e N=1, ho anche valutato altri indici di performance proprio di Rendering, in particolare, Number of Customers, Throughput e Response time, trovando il seguente risultato: il Number of Customers è stato di circa 70.36, il Throughput di circa 10.92, quindi il valore teorico del Response time sarebbe stato 6.44, molto vicino al risultato effettivamente trovato con la simulazione, pari a 6.43:



Number of Customers di Rendering con k=3 e N=1



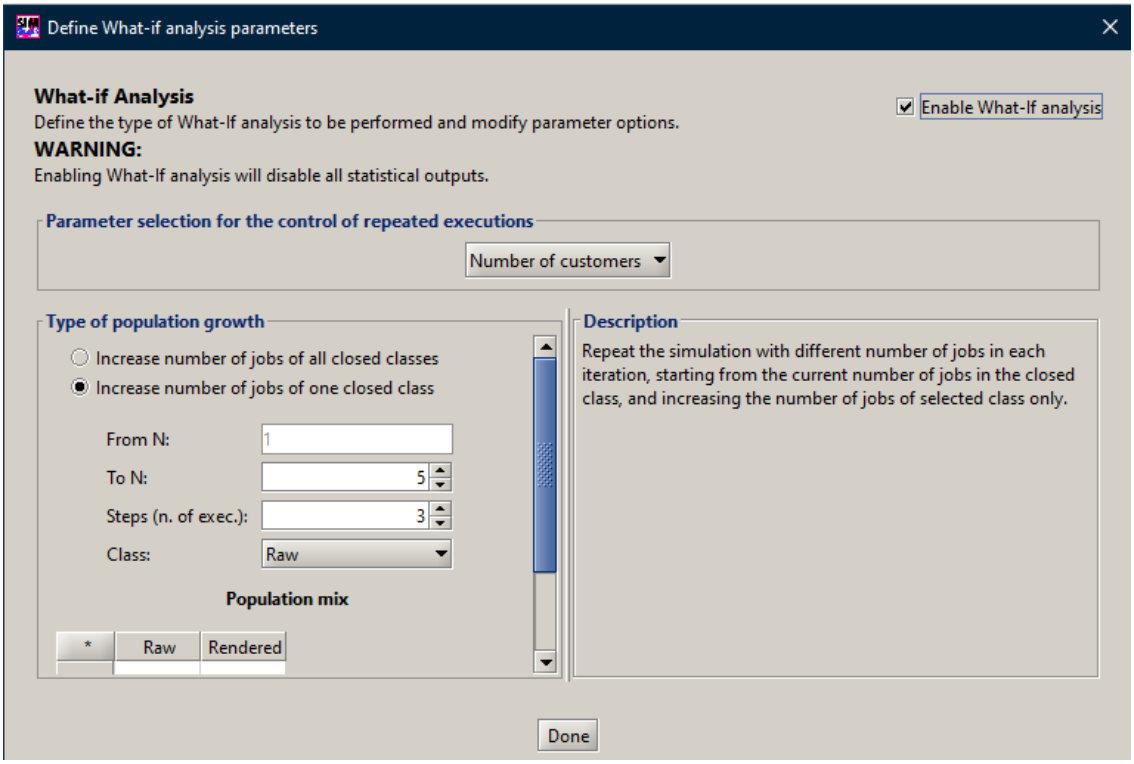
Throughput di Rendering con k=3 e N=1



Response time di Rendering con k=3 e N=1

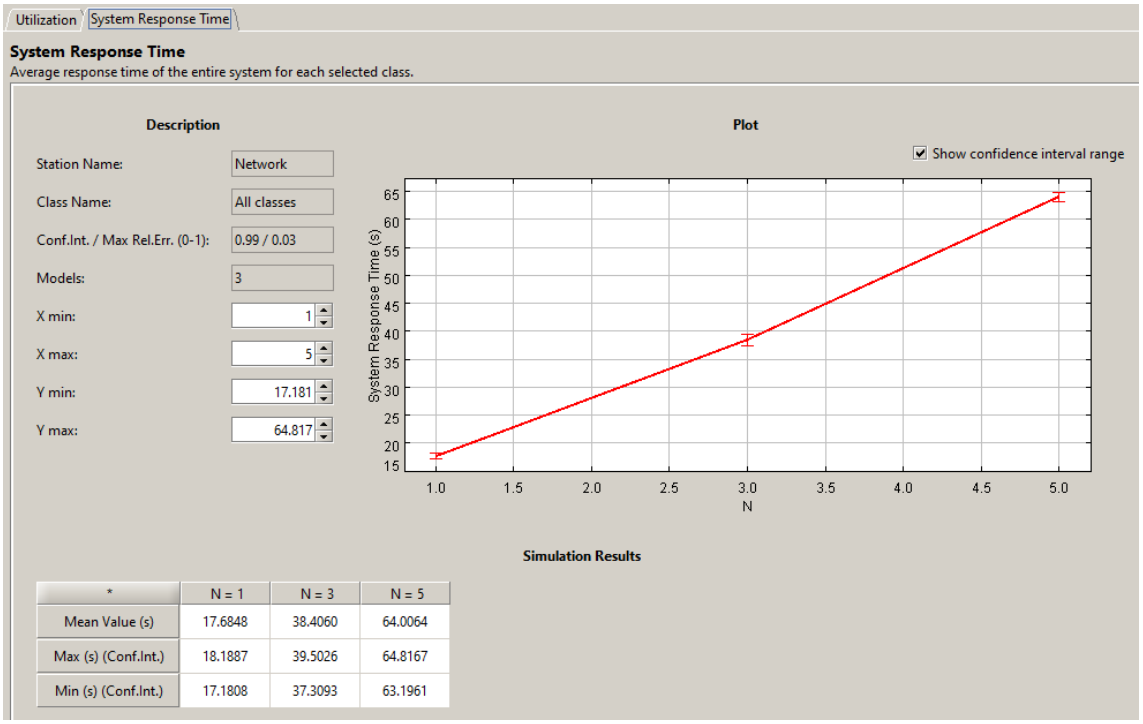
3.3 What-if analysis

Una volta appurato che con N=1, il minimo numero di server di Rendering tale da ottenere un Response time del sistema inferiore a 20 secondi fosse k=3, ho iniziato una What-if analysis, cambiando di volta in volta il numero k di server in Rendering, per valutare quale fosse il numero di processi N massimo, tale per cui il limite del Response time di cui sopra fosse effettivamente rispettato.



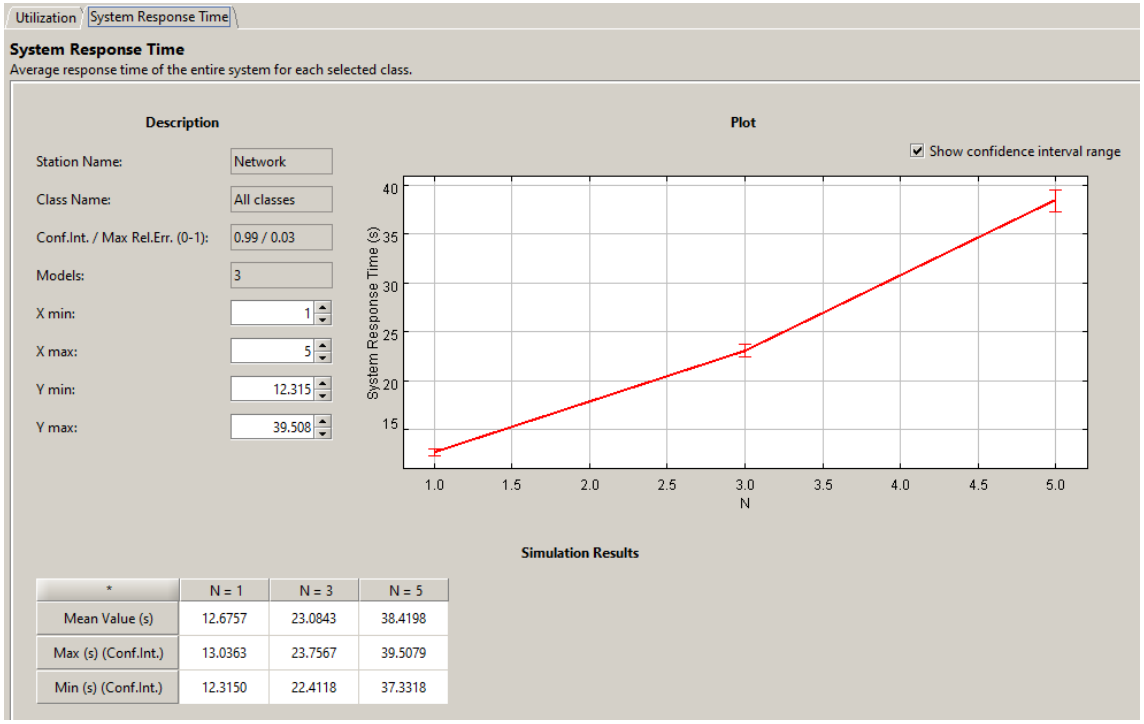
What-if analysis

Partendo da k=3, il risultato è stato che il Response time di sistema sotto i 20 secondi si poteva ottenere soltanto con N=1 (confermando anche la precedente analisi):

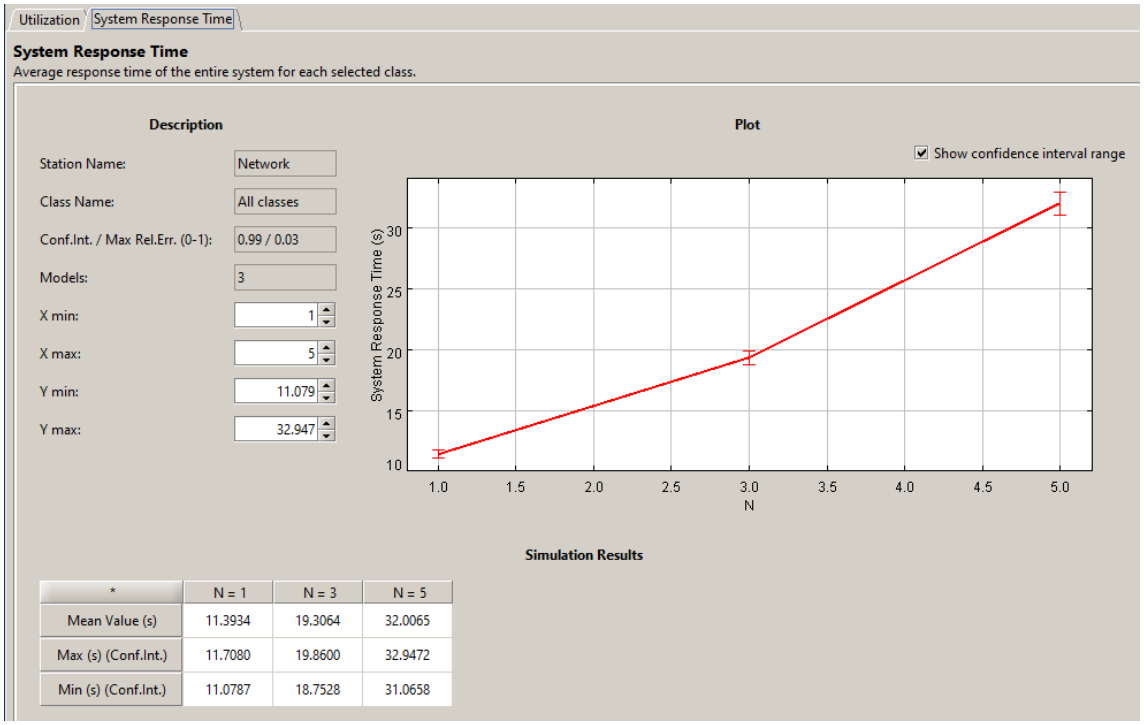


What-if analysis per k=3

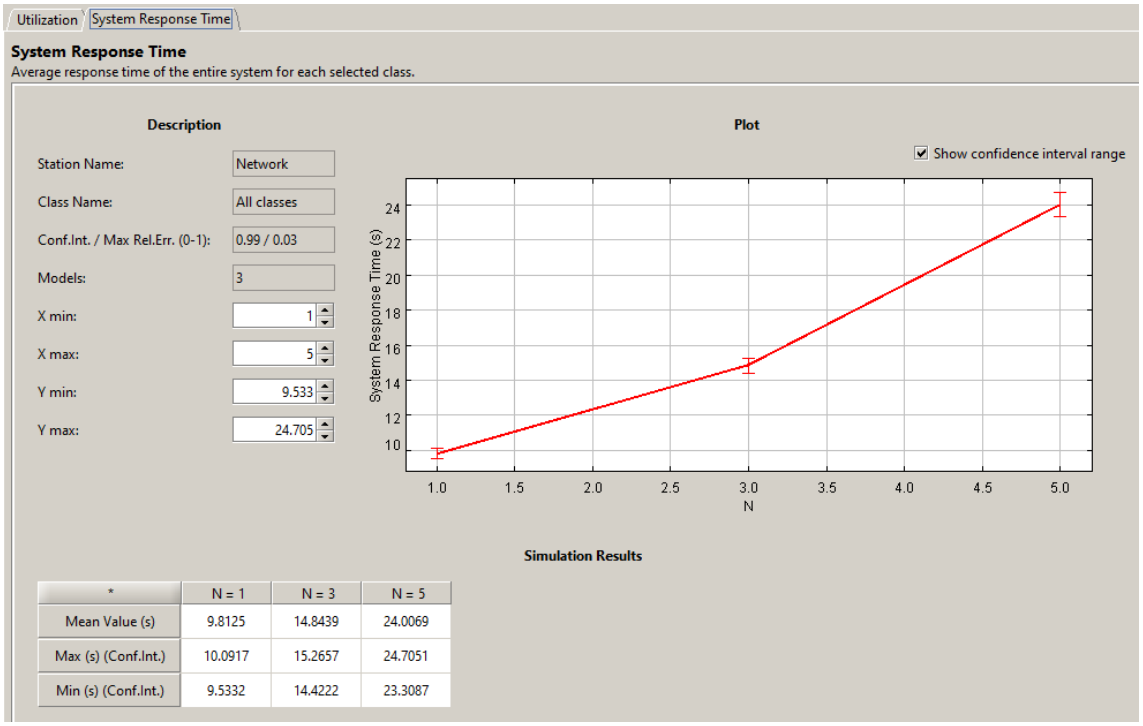
Quindi ho continuato l'analisi aumentando k e trovando i seguenti risultati (per N compreso tra 1 e 5):



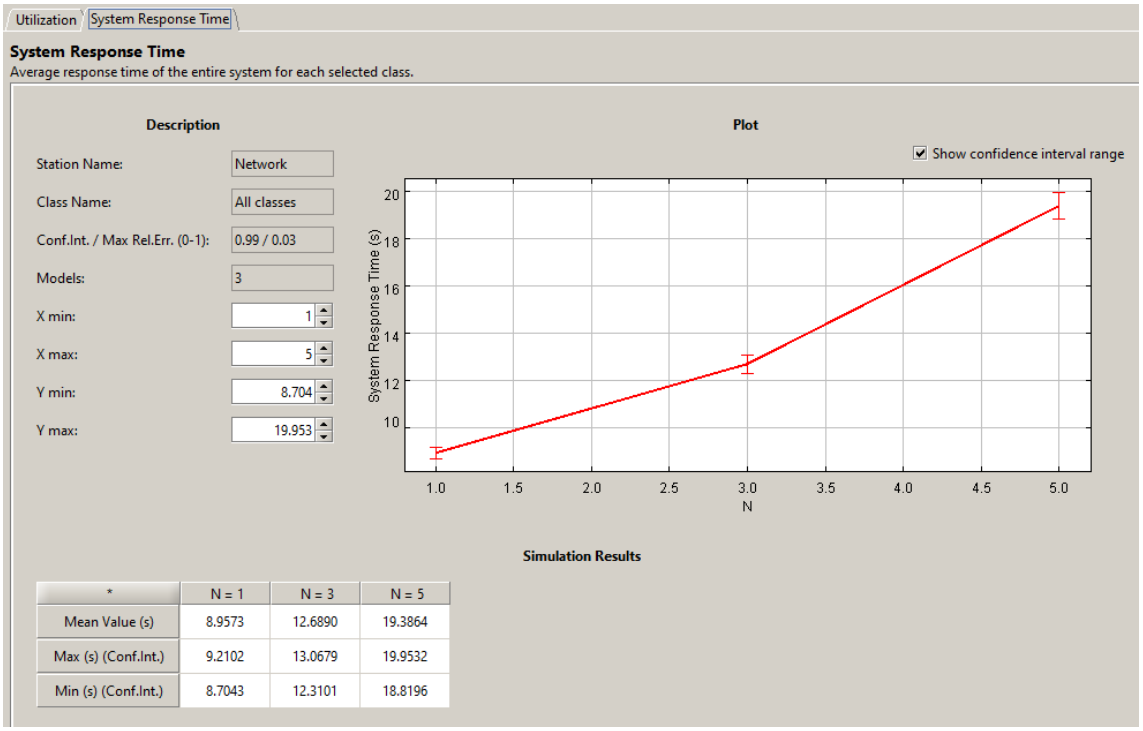
What-if analysis per k=5



What-if analysis per k=6



What-if analysis per k=8



What-if analysis per k=10

Ho riportato soltanto i risultati della What-if analysis con un numero di server k in Rendering che permettesse di aumentare il numero di processi N rispetto al k precedente, mentre gli altri risultati sono stati omessi per brevità.

Ricapitolando i risultati in una tabella (k_{min} è il numero minimo di server in Rendering, tale per cui, con N processi, il Response time di sistema fosse inferiore a 20 secondi):

N	$\mathbf{k_{min}}$
1	3
2	5
3	6
4	8
5	10
...	...

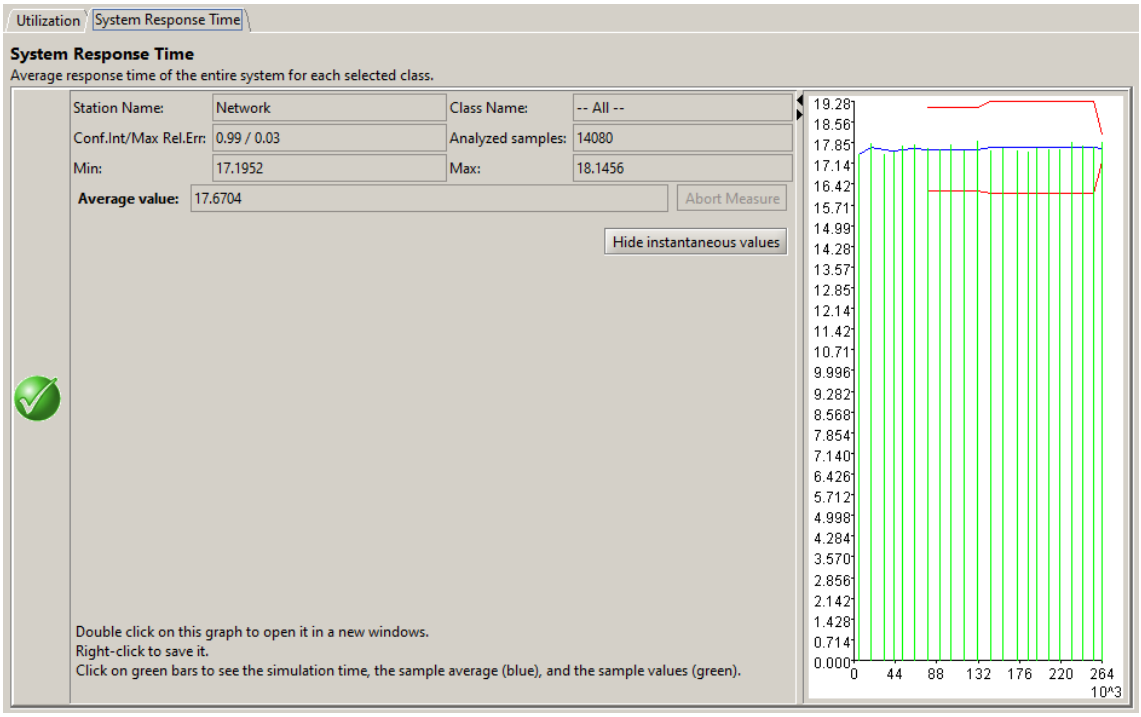
Di conseguenza, per garantire un Response time di sistema di 20 secondi al massimo, il miglior numero di processi è N=3, con il relativo k_{min}=6, perchè è l'unico "salto" (per numero di server k in Rendering bassi) che, aggiungendo un solo server, implica la possibilità di aggiungere un ulteriore processo nel sistema.

Questo, ad esempio, non è vero per il passaggio da 3 a 4 server in Rendering, in quanto il numero di processi accettabili sarebbe ancora solo uno, o, in altre parole, per poter passare da 1 a 2 processi, mantanendo il Response time del sistema sotto i 20 secondi, serve aggiungere almeno 2 server in Rendering.

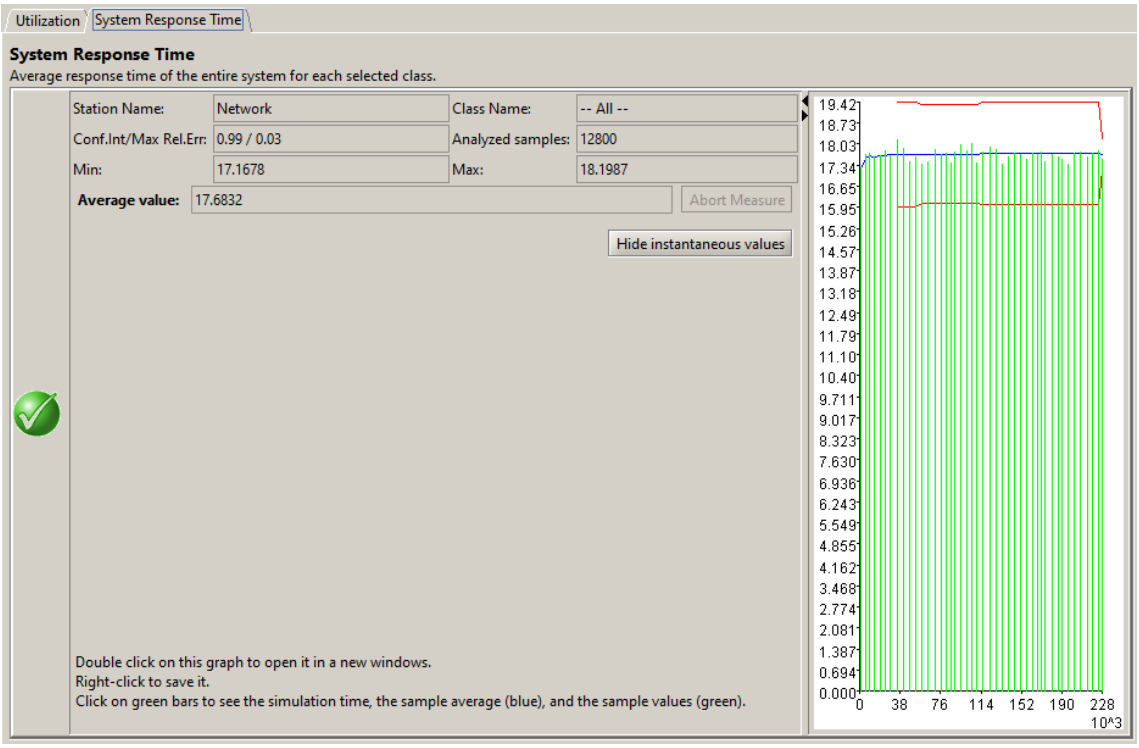
3.4 Uso di Processor Sharing

Nel modello che ho presentato fino a questo punto ho usato First Come First Served (FCFS) come queuing policy.

Per concludere l'analisi, ho, quindi, provato ad usare Processor Sharing (PS) come queuing policy per analizzare le eventuali differenze.
Con un solo processo nel sistema e k=3 server in Rendering, ovviamente il risultato in termini di Response time del sistema è praticamente identico:

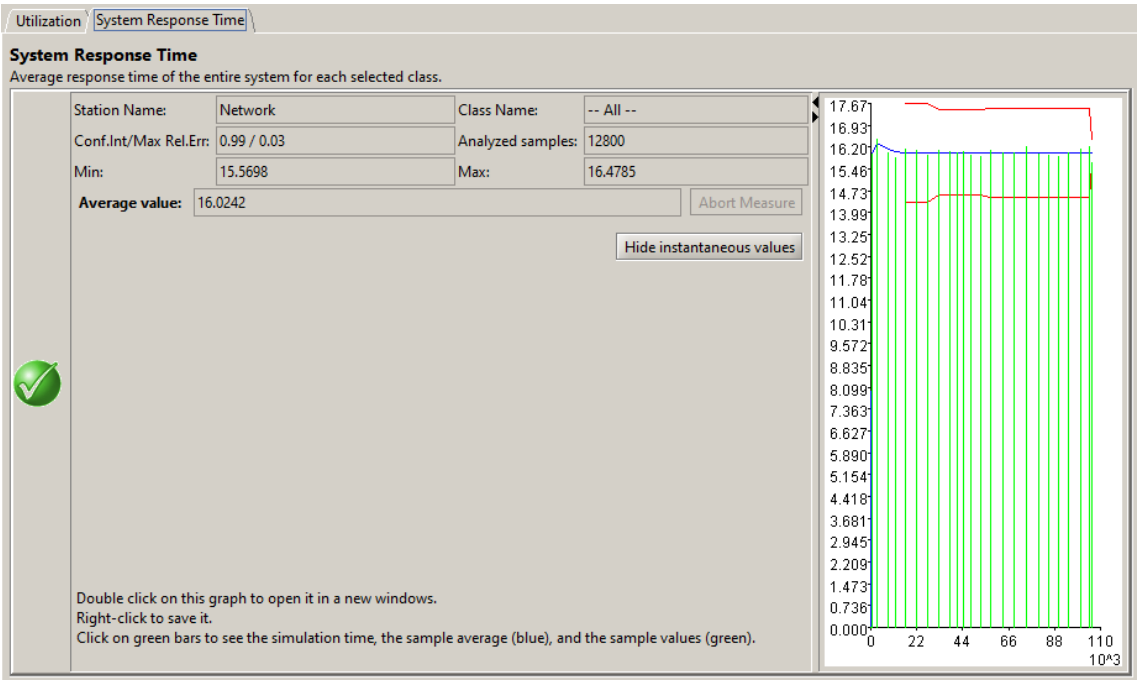


Sistema con FCFS, N=1 e k=3

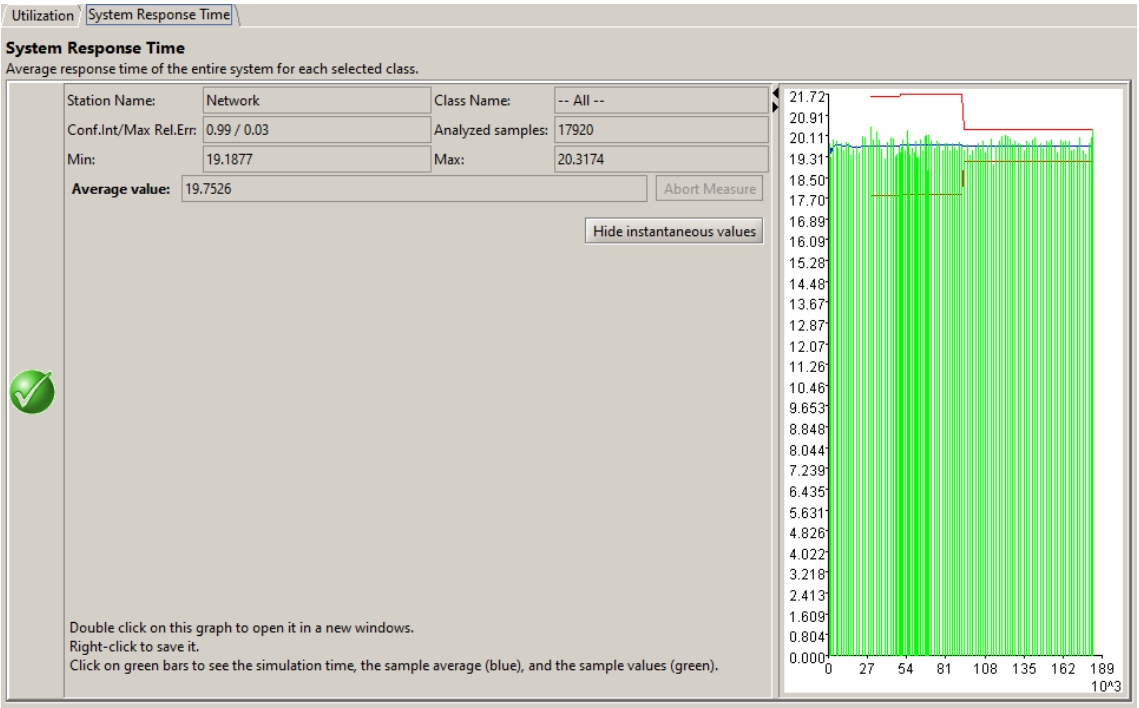


Sistema con PS, N=1 e k=3

Con 2 processi nel sistema, inizia già a notarsi una differenza in termini di un maggior Response time di sistema rispetto all'uso di FCFS, nonostante qui PS permetta comunque di mantenere k=5 se i processi sono due:

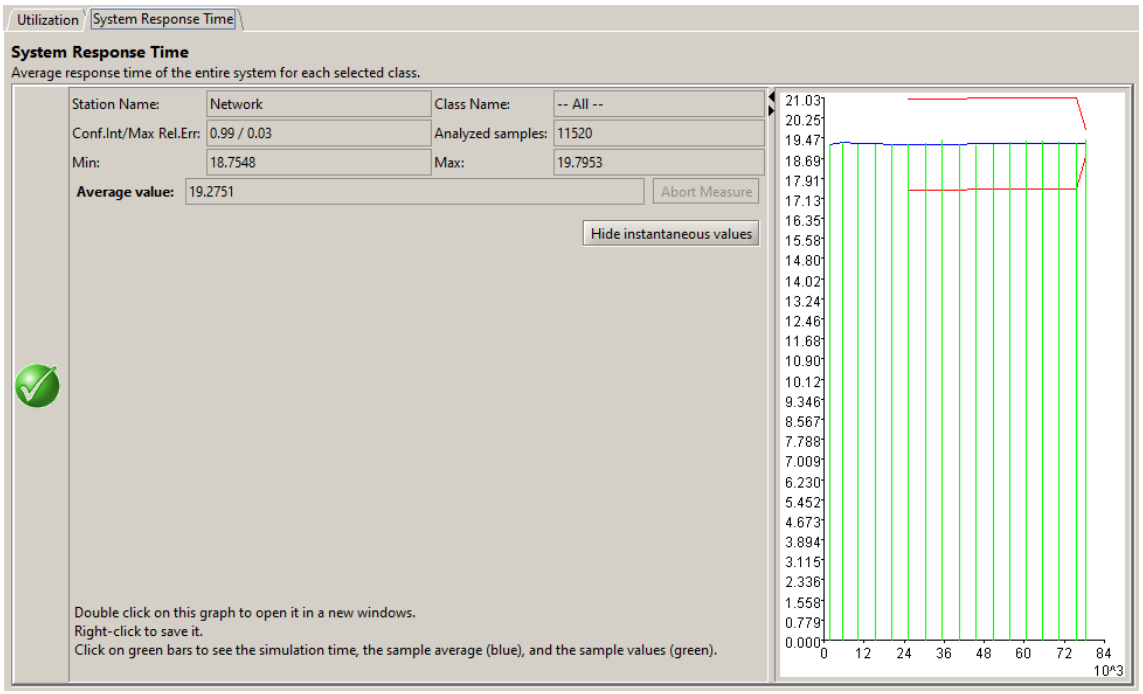


Sistema con FCFS, N=2 e k=5

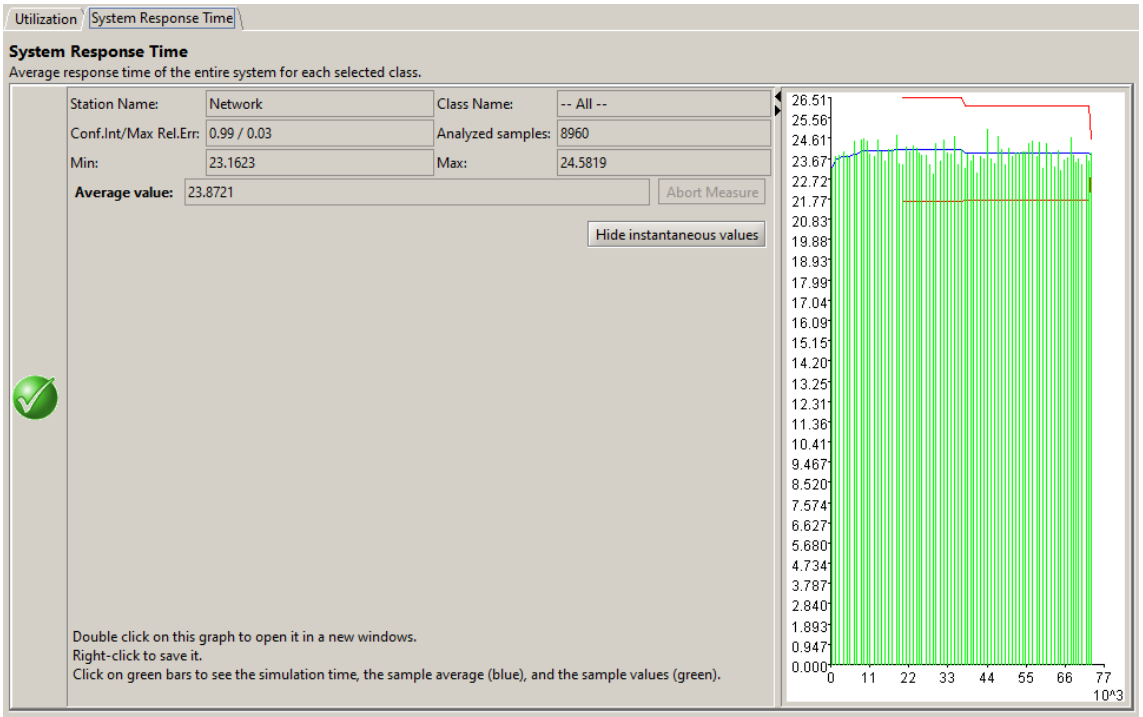


Sistema con PS, N=2 e k=5

Ma è con 3 processi nel sistema e k=6 che la differenza diventa davvero evidente:



Sistema con FCFS, N=3 e k=6



Sistema con PS, N=3 e k=6

In particolare, il Response time del sistema che con FCFS si manteneva al di sotto della soglia limite di 20 secondi, ora eccede questo limite.

In conclusione, dunque, in questo caso l'utilizzo di First Come First Served nelle code del modello è sicuramente da preferirsi all'utilizzo di Processor Sharing.