Chapter 3
Systems with Heterogeneous Workloads

3.1 Parameterization of Heterogeneous Workloads

As for the *characterization of the requests*, there is a direct correspondence between `Service times`, `Visits`, and `Service demands` used in *single-class* models and those used in *multiclass* models. However, since their values must be specified on a *per-class* basis, each parameter must be identified now with *two* indexes: the station and the class it will refers to. For example, the service demand D_r of resource r becomes now $D_{r,c}$, the `Service demand` of class-c request to resource r.

However, new problems arise when the growth of workload intensity has to be described as this can be done in different ways. Indeed, while to specify the workload intensity in single-class models is *sufficient* to know the `Number of customers` N_0 in execution in closed systems, or the arrival rate λ_0 and distribution of `Interarrival times` in open systems, the presence of multiple classes make these descriptions *no longer adequate*. Recall that with the index 0 (*zero*) of a metric we refer to the system as a *whole*.

In this section we first consider *closed* models, then we will analyze *open* models.

In closed models with of C *classes* of jobs, the workload intensity is described by the vector $\mathbf{N_0} = \{N_{0,1}, N_{0,2}, ..., N_{0,C}\}$ whose components are the number of jobs of each class in execution. The total number of jobs in execution is given by $N_0 = N_{0,1} + N_{0,2} + ... + N_{0,C}$. For example, $\mathbf{N_0} = \{25, 75\}$ means that in the closed model there are globally $N_0 = 100$ *jobs* in execution, 25 of class-1 and 75 of class-2.

A *new* parameter very useful for the description of multiclass workloads *growth* is the vector $\boldsymbol{\beta}$ representing the *fractions of jobs* of the C classes in execution in the system, that we will denote as *population mix* or *job-mix*:

$$\boldsymbol{\beta} = \{\beta_1, ..., \beta_C\} \quad \textit{with} \quad \beta_c = N_{0,c}/N_0 \quad \textit{and} \quad \beta_1 + \beta_2 + ... + \beta_C = 1 \tag{3.1}$$

Using the population mix, the workload $\mathbf{N_0}$ of the previous example can be described by $\mathbf{N} = N_0\,\boldsymbol{\beta}$, with $N_0 = 100$ and $\boldsymbol{\beta} = \{0.25, 0.75\}$.

G. Serazzi, *Performance Engineering*,
https://doi.org/10.1007/978-3-031-36763-2_3

The importance of the *job-mix* lies in the fact that in multiclass networks the ratio of the global utilizations of two stations is no longer *constant* with N_0, as in single class case, but depends on the fractions of jobs of the various classes in execution. Indeed, applying in *single-class* models the *Utilization law* to resources i and j we have: $U_i = X_0 D_i$ and $U_j = X_0 D_j$, and their ratio $U_i / U_j = D_i / D_j$ is *constant* with N. The immediate consequence of this behavior is that the *bottleneck* of the system may *migrate* among the resources as a function of the population mix. Thus, since it is known that the overall performance of a system is limited by the congested resource (i.e., the bottleneck), the fluctuation of the mixes may abruptly change them deeply.

While the definition of bottleneck is simple, i.e., the resource with the highest utilization, in multiclass models, the problem of bottleneck identification is not trivial since the same model can exhibit different bottlenecks depending on the population mix. Different types of bottlenecks can be identified. The *class-c bottleneck* is the station with the highest service demand of that class and saturates (its utilization tends to one) when the number of class-c customers grows to infinity. The problem in multiclass systems is that, as a function of the population mix, a station may saturate also if it is **not** a *class-bottleneck* (in this case the station will be referred to as *system-bottleneck* or *model-bottleneck*) or **more** stations may saturate *concurrently* with several mixes, referred to as *common saturation sector* (see [2, 3, 15]).

Therefore, to characterize the workload behavior in multiclass models, we must describe the variations of the mixes. In general, *different* $\boldsymbol{\beta}$ may yield *different* bottlenecks. Two types of workload increment should be considered: *proportional* and *unbalanced*. The population growth that consists of letting the total number of customers N_0 to grow keeping *constant* the population mix $\boldsymbol{\beta}$ is referred to as *proportional growth*.

According to this type of growth, in the example of workload above considered, the jobs in execution will be increased according to the proportions 25% of class-1 and 75% of class-2 since the mix is $\boldsymbol{\beta} = \{0.25, 0.75\}$. So, when the total number of jobs increases to 300, we will have 75 class-1 and 225 class-2 jobs in execution.

We have the *unbalanced population growth* when only one class of jobs, say c, increases. As $N_{0,c}$ continue to growth, the bottleneck of the system tends to the station that is the *class-c bottleneck*, the population mix tends to $\boldsymbol{\beta} = \{0, 0, 0, ..., 1_C\}$, and the performance tend to the asymptotes of single-class workloads.

To support users who need to model the different types of population growth, JMT implement specific features of the `What-if` analysis that allow the automatic increment of the population of a single class only (see, e.g., Fig. 3.2) or the generation of all the possible mixes of two classes in closed models (see, e.g., Fig. 3.7b).

Most of what has been previously described for the closed models also applies to *open models*. The number of jobs in execution of the various classes must be replaced with the corresponding arrival rates. So, the global arrival rate to a open system is $\boldsymbol{\lambda_0} = \{\lambda_{0,1}, \lambda_{0,2}, ..., \lambda_{0,C}\}$ and the population mix is described by:

$$\boldsymbol{\beta} = \{\beta_1, ..., \beta_C\} \qquad \beta_c = \lambda_{0,c}/\lambda_0 \qquad \beta_1 + \beta_2 + ... + \beta_C = 1 \tag{3.2}$$

Differences between open and closed systems lie in the *bottleneck switch* behavior. In the former, the bottleneck migrate *instantaneously* between two resources without going through a *common saturation section*, i.e., the set of mixes that saturate both concurrently. Regarding the *scheduling disciplines* of the various classes in multiclass models, there are differences as a function of the solution technique adopted. While with *simulation* the users have practically *no limitations* (some minor incompatibilities may take place as a function of the types of discipline selected), the analytical technique introduce some constraints. Typically, the queueing networks that are solved analytically are of the *separable* types (see, e.g., [9, 36]) and their solution (the stationary probability of their states) can be obtained by the product of the individual solutions of the stations. The computational complexity introduced by the presence of multiple classes that may have different scheduling algorithms, and usually by the large numbers of stations and customers has required the introduction of some limitations. An important theorem, the `BCMP` (see, e.g., [6, 36]), for *open, closed,* and *mixed* queueing networks define the characteristics that a multiclass network should have to be separable and thus to be solved analytically with efficient algorithms. Each station must be of one of the following types:

- a *queue* station with *FCFS scheduling discipline* (requests are served according to the sequence of arrival), with one or more servers, having the *same* exponential distribution of `Service times` for all the classes. For each resource, the `Service times` $S_{r,c}$ must be the *same* for *all* the classes. The differences between the classes may be considered using the number of *visits* $V_{r,c}$ to the resources, providing the possibility to have different `Service demands` $D_{r,c}$ for the same station (which in any case *cannot* be modeled with FCFS discipline).
- a *queue* station with PS (*processor-sharing*) scheduling discipline: the n requests in the station are served simultaneously receiving each $1/n$ of the server capacity. For example, in a queue station with one server if during the execution of a request that has `Service time` S = 2 s there are 10 requests in the station, then the execution of this request will be completed after 20 sec. This discipline is commonly used to model the time quantum of the processors, in this case the quantum tends to zero. The distribution of the service times, and their *means*, can be *general* and *different* for *each class.*
- a *queue* station with LCFS-PR (*last-come first-served preemptive-resume*) scheduling discipline. When a new request arrives, it interrupts the execution currently in progress and starts its execution immediately. When it is completed, the last preempted request resumes the execution at the point it was interrupted. The distribution of the `Service times`, and their *means*, can be *general* and *different* for *each class.*
- a *delay* station, referred to as IS *infinite servers* station, in which each request has its dedicated server. Its `Response time` coincide with the `Service time` since there is no queue time. The distribution of the service times, and their *means*, can be *general* and *different* for *each class.*

Load-dependent service times are allowed. For PS, LCFS-PR, and IS stations the service times for the requests of a class may depend on the number of requests of that class in the station, or on the global number of requests in that station. For FCFS station the service times may depend only on the global number of requests of all the classes in that station.

In conclusion, users *must be sure* that the multiclass models who want to solve *analytically* have stations of the four types described. For example, if you want to solve analitycally a model that has different per-class `Service times` $S_{r,c}$ on the same resource, the JMVA will solve it in any case (it compute the $D_{r,c}$) but you should be aware that the modeled scheduling discipline is PS and not FCFS. If in any case it is necessary to solve this model with the FCFS discipline, then there is no other choice than to use simulation.

3.2 Motivating Example of Multiclass Models

tags: closed, two-class, Delay/Queue, JMVA.

In this section we describe an example *purposely designed* to emphasize the errors on the performance forecast that can be obtained from the *same model* assuming that a multiclass workload consists of a single class of jobs. The counterintuitive behavior exhibited by some performance indexes of multiclass models as a function of the fluctuations of the classes of jobs in execution is also investigated. We solve this model analytically with JMVA.

3.2.1 Problem Description

Consider a powerful web server, accessed by administrative staff and graduate students of a university, which has two main resources: *CPU* and *Storage* (Fig. 3.1a). The workload consists of *two* different applications. The first one is used to manage the *administrative* procedures concerning the students curricula (tuition fees payments, courses attended, grades obtained, ...). The second one is devoted to the management (uploads, downloads, folder structure) of the course materials/*documents* (slides, notes, class exercises, homeworks, exams) that professors, assistants, and students access. According to the resource requirements of the two types of users, *two* classes of jobs, referred to as *Adm* and *Doc*, are identified.

Initially, we want to investigate the effects on performance indexes generated by different workload scenarios. More precisely, we increment the number of jobs of *one class only* while keeping constant the jobs of the other class. The different growth rates of the various classes of the workload are described by modifying the parameter $\boldsymbol{\beta}$. To model the growth of class-i jobs, we increase the corresponding β_i.

Then, we want to illustrate the errors that can be introduced in the performance forecast by a wrong assumption on the number of classes of the workload. We solve the same system model assuming the two-class workload as consisting of a

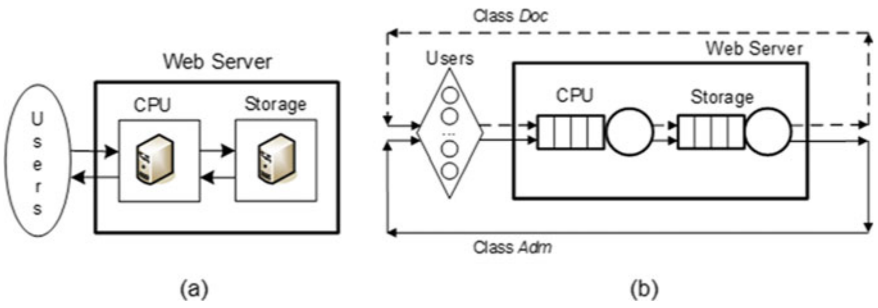


Fig. 3.1 Web server layout (**a**) and closed queueing network with two different applications (**b**)

single-class of jobs. We consider a `base` system and an `upgraded` system with a CPU more powerful of a factor of five (the corresponding CPU service demands are decreased by five times). The behavior of the performance indexes are studied with respect to all the possible combinations of the two classes of jobs in execution, i.e., all the possible population mix.

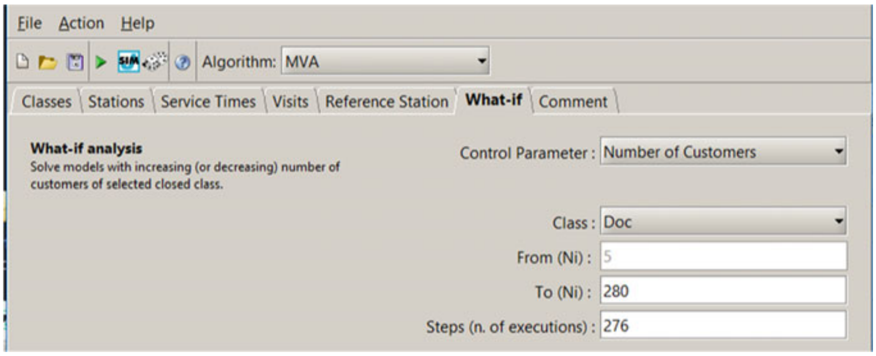
3.2.2 Model Implementation

We use a *closed* model (Fig. 3.1b) since the customers that have access to the server are limited (administrative staff and graduate students). It consists of three resources: `Users`, `CPU`, and `Storage`. The `Users` station is of *delay* type.

The two classes of the workload are characterized by the `Service demands` shown in Table 3.1. Let N_0 be the global number of jobs of the two classes. To model the growth of class-*Doc* jobs only (*unbalanced* population growth), we use the `What-if` feature with `Number of customers` and class-*Doc* as `control parameters` (Fig. 3.2). The $N_{0,Doc}$ values range from 5 to 280 with step of 5. To study the effects on performance forecast corresponding to the two different assumptions on the number of classes (*one* and *two*) in the workload, we consider two configurations, i.e., `base` and `upgraded`, of the same system. The behavior of performance indexes in the two configurations has been investigated modeling

Table 3.1 `Service demands` [s] of the *two classes* of jobs

Resources (stations)	Two classes	
	Adm	*Doc*
`Users think time`	3	10
`CPU`	0.20	0.100
`Storage`	0.050	0.60



What-if analysis
Solve models with increasing (or decreasing) number of customers of selected closed class.

Control Parameter : Number of Customers
Class : Doc
From (Ni) : 5
To (Ni) : 280
Steps (n. of executions) : 276

Fig. 3.2 Parameters of the `What-if` for the description of the *unbalanced* population growth: only class-*Doc* jobs increase while class-*Adm* jobs are kept constant

all possible population mixes $\boldsymbol{\beta}$ with $N_0 = 300$ jobs (Fig. 3.5) and modifying the `Service demands` of Table 3.1 (see Table 3.2).

3.2.3 Results

In what follows we will describe the operations required to achieve the *objectives* of the study (referred to as *Obj.1–Obj.2*).

Obj.1: Show the counterintuitive result that with a multiclass workload the Global System Throughput X_0 can decrease in spite that the global number N_0 of jobs in execution increases
We consider the model with the two-class workload whose service demands are shown in Table 3.1. The initial workload is $\mathbf{N} = \{20, 5\}$, globally $N_0 = 25$ jobs are in execution, 20 of class-*Adm* and 5 of class-*Doc*. The volume of traffic due to the class-*Doc* jobs is expected to increase during the next semester up to 280. This behavior is the typical *unbalanced population growth* that can be used when one class increases more than the others. We use the `What-if` feature of JMVA to evaluate the `Global System Throughput` X_0. The parameters that describe the increase of class *Doc* jobs are shown in Fig. 3.2. The `Control Parameter` is the `Number of customers` of class-`Doc` $N_{0,Doc}$, and the execution of 276 models with its increasing values from 5 to 280 are required.

In Fig. 3.3a the behavior of the `Global` and `per-class System Throughput` are shown for the number of jobs N_0 in execution increasing from 25, $\mathbf{N} = \{20, 5\}$, to 300, $\mathbf{N} = \{20, 280\}$. Initially X_0 increases until the number of *Doc* jobs reaches 19, corresponding to the maximum value of $X_0 = 5.416$ j/s. Then, any further increase of $N_{0,Doc}$, and clearly of N_0, corresponds to a decrease of X_0 (with $N_{0,Doc} = 280$ it is $X_0 = 2.724$ j/s). *How is it possible this happens*?

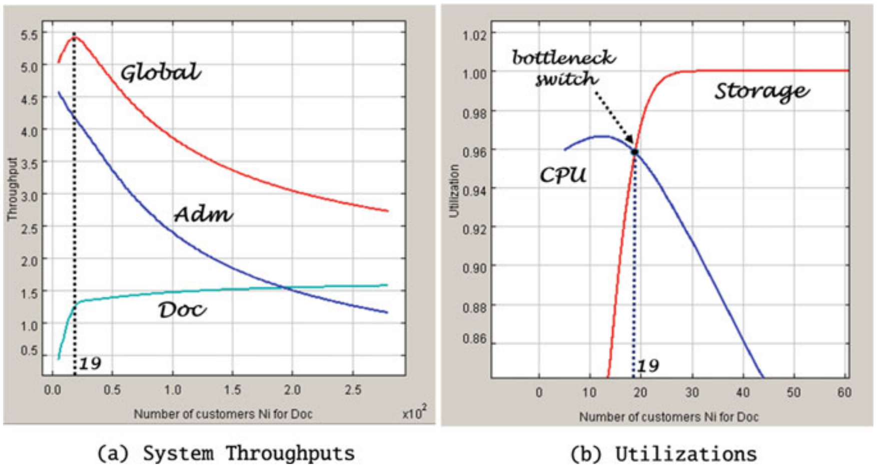


Fig. 3.3 *Counterintuitive* behavior of performance indexes (Base system) with *unbalanced* increase of the Global Number N_0 of jobs in execution: only class-*Doc* jobs increases from 5 to 280

The answer is prompted by Fig. 3.3b showing the Global Utilization of CPU and Storage. For $N_{0,Doc} \leq 18$ the CPU is the most utilized resource, while for $N_{0,Doc} \geq 19$ the Storage is the most utilized. We are addressing the *bottleneck switch* phenomenon that can occur with multiclass workloads when different classes have their highest service demands on different stations. The basic concept is the following: the service demands of the various classes at the bottleneck station determine the performance of the global system. Since when the station bottleneck changes typically also the corresponding service demands are different, this migration may have a deep impact on the performance. While the *identification of the bottleneck* in single-class models is easy, in multiclass models is more complex [3]. As described in Sect. 3.1, with multiclass workloads the bottleneck may migrate among stations as a function of the percentage of jobs of the different classes in execution, i.e., of the *population mix*.

With the workload behavior considered in this *Obj.1* study, only class-*Doc* jobs increase from 5 to 280 and the population mix range from $\boldsymbol{\beta} = \{0.8, 0.2\}$ ($\mathbf{N} = \{20, 5\}$) to $\boldsymbol{\beta} = \{0.066, 0.933\}$ ($\mathbf{N} = \{20, 280\}$). When the *Doc* jobs are ≤ 18, the contribute of the *Adm* jobs, with $D_{max,Adm} = 0.2$ on CPU, is fundamental for the saturation of CPU. When *Doc* jobs, with $D_{max,Doc} = 0.6$, are ≥ 19 the load generated on Storage makes its global utilization predominant with that of the CPU.

As the number of *Doc* jobs continue to increase, asymptotically it will be ($N_{0,Doc} \to \infty$), and the workload assume the characteristics of a *single-class* with $\boldsymbol{\beta} \to \{0, 1\}$. In this case, the maximum system throughput is given by $1/D_{max,Doc}$. In our workload the max Service demand of class-*Doc* is $D_{Sto,Doc} = 0.6$ s, thus it will be $X_0^\infty = 1/D_{Sto,Doc} = 1.666$ j/s. As can be seen from Figs. 3.3a, 3.4a, the System throughput of class-Doc and of the Global system tend to this

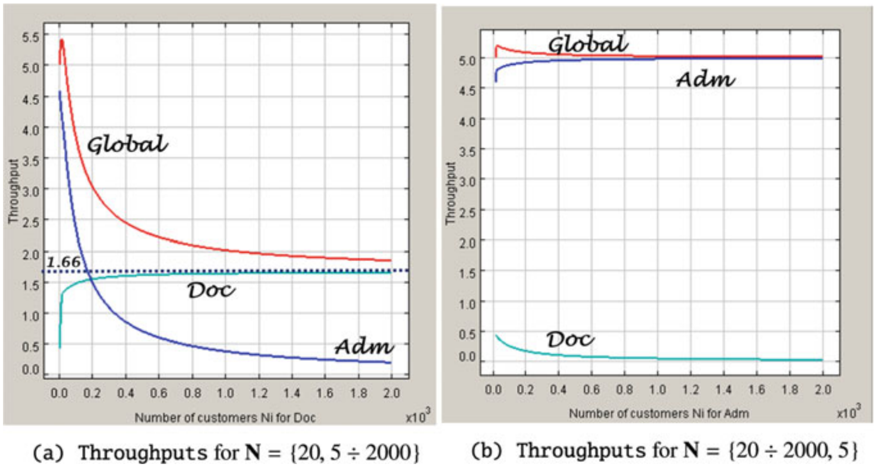


Fig. 3.4 `System throughput` asymptotes of the (`Base` system) when only class-*Doc* jobs increase from 5 to 2000 (**a**), and only class-*Adm* jobs increase from 20 to 2000 (**b**)

value: for $N_{0,Doc} = 280$ *jobs* it is $X_{0,Doc} = 1.57$ job/s and for $N_{0,Doc} = 2000$ *jobs* it is $X_{0,Doc} = 1.65$ job/s. The `Global System Throughput` X_0 decreases as $N_{0,Doc}$ increases beyond 19 jobs.

To emphasize the impact of the population mix on the `Global System Throughput` we ran two experiments with unbalanced population growth. Starting from the same initial workload $\mathbf{N} = \{20, 5\}$, in the first one we increase to 2000 only class-`Doc` jobs while in the second one we increase to 2000 only class-`Adm` jobs. The `System throughputs` are shown in Fig. 3.4.

In Fig. 3.4a $X_{0,Doc}$ and X_0 tend to the same asymptotic value 1.666 *j/s* while in Fig. 3.4b $X_{0,Adm}$ and X_0 tend to the same asymptotic value 5 *job/sec*. The `Global System Throughput` X_0 tend to $X_{0,Doc}$ in (a) and to $X_{0,Adm}$ in (b). The differences between the two asymptotic values are evident. It should be pointed out that these values are *not* bounds! Indeed, a program mix that maximize the `Utilization` of *all* the resources of a system (see Fig. 3.3) maximize also the `System throughput`.

Obj.2: Show that assuming a multiclass workload as single-class allows the construction of models that generate very inaccurate performance forecast. Some counterintuitive results (other than those of Obj.1) that occur with multiclass models are also shown.

In this study (inspired, with some differences, by [25]) we will show that the performance projections obtained using a *wrong assumption* for the workload characterization, i.e., the workload is assumed to consist of a single class instead of multiple classes of customers, are *unreliable*.

We consider the closed system with three stations (Fig. 3.1b), that process the two-class workload whose `Service demands` are shown in Table 3.1.

Table 3.2 Inputs and outputs of the *single-* and *two-class* models, for the original (`Base`) and the upgraded (`Up`) systems. The two-class workload is $\mathbf{N} = \{255, 45\}$ jobs, $\boldsymbol{\beta} = \{0.85, 0.15\}$

		Single-class workload			*Two-class workload*			
		`Aggregate`			`Adm`		`Doc`	
		`Base`	`Up` wrong	`Up` correct	`Base`	`Up`	`Base`	`Up`
Inputs	D_{CPU}	0.180	0.036	0.039	0.2	0.04	0.1	0.02
	D_{Sto}	0.159	0.159	0.059	0.05	0.05	0.6	0.6
	Z_0	4.390	4.390	3.118	3	3	10	10
	R_0	49.649	$\boxed{43.38}$	$\boxed{14.683}$	$\boxed{54.322}$	$\boxed{12.393}$	$\boxed{\mathbf{30.801}}$	$\boxed{\mathbf{147.228}}$
Output	X_0	5.551	$\boxed{6.279}$	$\boxed{16.857}$	4.448	16.565	1.102	0.2862
Measures	U_{CPU}	1	$\boxed{0.2262}$	$\boxed{0.6683}$	0.8897	0.6626	0.1103	0.0057
	U_{Sto}	0.8842	1	1	0.2224	0.8283	0.6617	0.1717

In what follows we will refer to the values reported in Table 3.2 obtained with the two-class workload $\mathbf{N} = \{255, 45\}$ jobs, the corresponding population mix is $\boldsymbol{\beta} = \{0.85, 0.15\}$. The study has been carried out according to the following steps:

step (1)—First, we assume to know that the workload consists of $N_0 = 300$ jobs belonging to two classes *Adm* and *Doc* whose service demands are shown in Table 3.1. We consider the population mix $\boldsymbol{\beta} = \{0.85, 0.15\}$, i.e., the workload is $\mathbf{N} = \{255, 45\}$ jobs, 255 *Adm* and 45 *Doc*. Some output measures (`System Response time` R_0, `System Throughput` X_0, and the `Utilizations` of CPU and Storage) obtained from the execution of the two-class model are shown in columns `Adm-Base` and `Doc-Base` of Table 3.2.

step (2)—From the outputs of the two class model we compute the correspondent *single class* aggregate model (column `aggregate-Base`). The aggregate values of `Utilization` and `System Throughput` X_0 are obtained summing the correspondent per-class indexes: $U_{CPU} = U_{CPU,Adm} + U_{CPU,Doc} = 0.89 + 0.11 = 1$ $U_{Sto} = U_{Sto,Adm} + U_{Sto,Doc} = 0.22 + 0.66 = 0.88 X_0 = X_{0,Adm} + X_{0,Doc} = 4.448$ $+ 1.102 = 5.551$. For the `System Response time` R_0, according to the Little law $N_i = X_i R_i$, the per-class values must be weighted by the relative throughput:

$$R_0^{Base} = \frac{N_0}{X_0} = R_{0,Adm}\,\frac{X_{0,Adm}}{X_0} + R_{0,Doc}\,\frac{X_{0,Doc}}{X_0} = 49.649\,sec \qquad (3.3)$$

This is the correct computation of the `System Response time` with *multiclass workloads*.The same weights must be applied to the computation of the aggregate service demands $D_{CPU}^{Base} = 0.180\,s$ and $D_{Sto}^{Base} = 0.159\,s$. The weights of the relative throughputs have also the following intuitive interpretation. The number of jobs of the two classes *Adm* and *Doc* executed in the interval T are $C_{0,Adm}$ and $C_{0,Doc}$, respectively. This means that in the log file of the system executing the two classes workload there will be $C_{0,Adm}$ times the value of $R_{0,Adm}$ and $C_{0,Doc}$ times the value of $R_{0,Doc}$. Thus, to compute the mean of all the $R_0 s$ we need to weight the two values with the respective times they appear in the file. And, dividing by T both the terms of the ratios $C_{0,Adm}/C_0$ and $C_{0,Doc}/C_0$ we obtain $X_{0,Adm}/X_0$ and $X_{0,Doc}/X_0$, that are the weights considered in Eq. 3.3.

step (3)—Now consider a new analyst who does not know anything about the system workload and builds a single-class model considering all measures of the log file as belonging to a single type of jobs. If he made the right computations, he will get the same values shown in the column `Aggregate-Base` for both input parameters and output measures. We note that these values are the same as those already computed in the previous step. Indeed, he, without being aware, automatically applies for their computation the correct weights described in **step 2**. At this point, the analyst has the *wrong certainty* that the implemented model is *correct*!

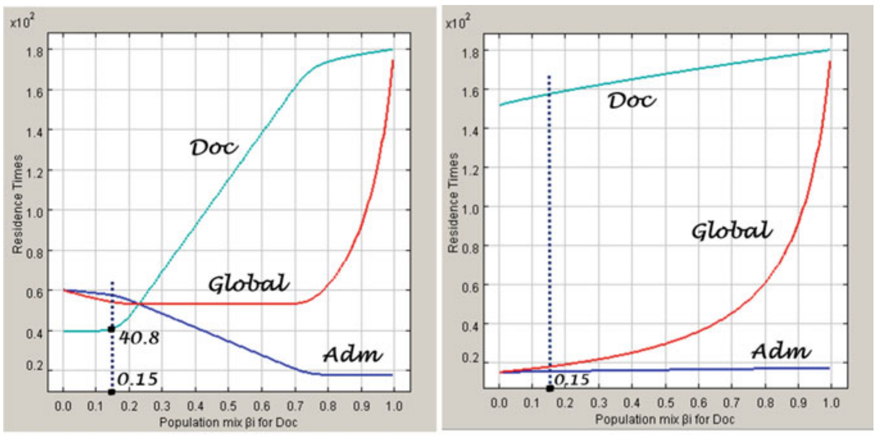
step (4)—Now we will use the two workload models (the one with two classes and the one with a single class) for the performance projections. An increase of *Doc* customers is expected in the near future. We want to evaluate the effect on the response time $R_{0,Doc}$ of *Doc* jobs that will be generated by an increase of the CPU speed. We assume to consider a multicore processor that for the workload considered increases by a factor of five the CPU speed.

The primary effect of this upgrade is the decrease of the CPU service demands $D_{CPU,Adm}$ from 0.2 to 0.04 and of $D_{CPU,Doc}$ from 0.1 to 0.02 (see D_{CPU} in columns `Adm-Up` and `Doc-Up`, respectively). The execution of the two-class model compute the indexes reported in the lower part of these columns. Applying the computations described in **step 2)** it is possible to derive the correct values of the aggregated single-class model corresponding to the two-class model (see column `Aggregate-Up correct`).

Analyzing the `System Response times` of the two classes, we may see that while the $R_{0,Adm}$ decreases of 77%, the $R_{0,Doc}$ **increases of 377%** (from 30.8 to 147.2 s)! This is a **counterintuitive result: performance degrades with a CPU upgrade!** The motivation of this result hampered by intuition is related to the *switch* of the bottleneck from the CPU to the storage which take place after the upgrade. Indeed, since the throughput $X_{0,Adm}$ of class-*Adm* (that has the CPU as class-bottleneck) increases from 4.4 to 16.5 j/s, the competition for the *Storage* increases a lot (U_{Sto} reach saturation) and the class-*Doc* jobs, that are heavily storage bound (it is $D_{Sto,Doc} = 0.6\ s$, the Storage is the class-*Doc* bottleneck), experience a *strong degradation* of the response time.

Figure 3.5 shows the behavior of the `System Response times` of the two classes and the aggregated value as a function of all the population mixes, of the original system (a) and the upgraded version (b). The workload consists of 300 jobs, ranging from $\mathbf{N} = \{300, 0\}$ to $\mathbf{N} = \{0, 300\}$ jobs. The differences between the behavior of the correspondent curves are evident.

step (5)—We consider here the single-class model built in **step 3)** by the ignorant analyst, assuming that all the jobs as belonging to the same single class. The effect of the CPU upgrade is a decrease in service demand D_{CPU} from 0.180 to 0.036 s,



(a) SRT *vs* program mixes of Base system (b) SRT *vs* program mixes of Up system

Fig. 3.5 `System Response times` *versus* all $\boldsymbol{\beta}$ population mixes of the *original* (`Base`) and the *upgraded* (`Up`) systems, with 300 jobs. Dashed lines represent $\boldsymbol{\beta} = \{0.85, 0.15\}$ used in Table 3.2

see columns (`Aggregate-Base`) and (`Aggregate-Up wrong`). The output measures (Table 3.2) obtained from the execution of this single class model with $N = 300$ *jobs* show an improvement of performance: R_0 *decreases* from 49.64 to 43.38 *s*, and X_0 *increases* from 5.5 to 6.2 *j/s*. As this qualitative behavior corresponds to the expectations, the analyst my have the wrong impression that the implemented model is correct. Indeed, comparing the values of the two columns `Aggregate-Up wrong` and `Aggregate-Up correct` it is possible to understand immediately the **large errors** affecting R_0 and X_0: -66% and +168%, respectively. Thus, we may conclude that

the performance forecast based on single-class models of heterogeneous workloads are inaccurate when used to obtain projections for the average aggregate job. Furthermore, as Fig. 3.5 shows, it is evident that the average aggregate job cannot be used to derive reliable projections for each class of the two-class workload.

Let us remark that the per-class `System Response times` plotted by `JMVA` in Fig. 3.5 are the sum of the `Residence times` of a job of that class at *all* the resources of the system, *including* the `Reference station`. Thus, for example, we should add $Z_{0,Doc} = 10$ s (the `Residence time` of `Doc` jobs at the `Reference station`) to the `System Response time` of `Doc` jobs $R_{0,Doc} = 30.801$ s of Table 3.2 to obtain the value 40.801 plotted in Fig. 3.5a in correspondence to program mix $\boldsymbol{\beta} = \{0.85, 0.15\}$.

3.3 Performance Optimization of a Data Center

tags: closed, two-class, Queue, JMVA.

In this case study we illustrate a general approach applicable to several capacity planning problems with an example based on the performance optimization of a data center with a workload consisting of *heterogeneous* applications, [14, 38]. The problems of bottleneck identification and migration are addressed as a function of the fluctuations of the different types (*classes*) of requests being executed. The topic of *load balancing* is also investigated.

3.3.1 Problem Description

A data center partition consists of six servers utilized by business critical applications that access to sensitive data stored. The area is highly protected for both physical access and digital security. It consists of one `Web Server`, two `Application Servers`, and three `Storage Servers` (see Fig. 3.6). The access to the applications and data stored on these servers is permitted only to a limited number of employees with the appropriate authorization. Based on the different requirements, in terms of amount of resources used and Quality of Service (QoS) targets, *two* types of applications can be identified in the workload. The *two classes* of requests, called class-1 and class-2, generated by the two applications are focused on business logic processing the first, and on intensive data processing (search, update) the second.

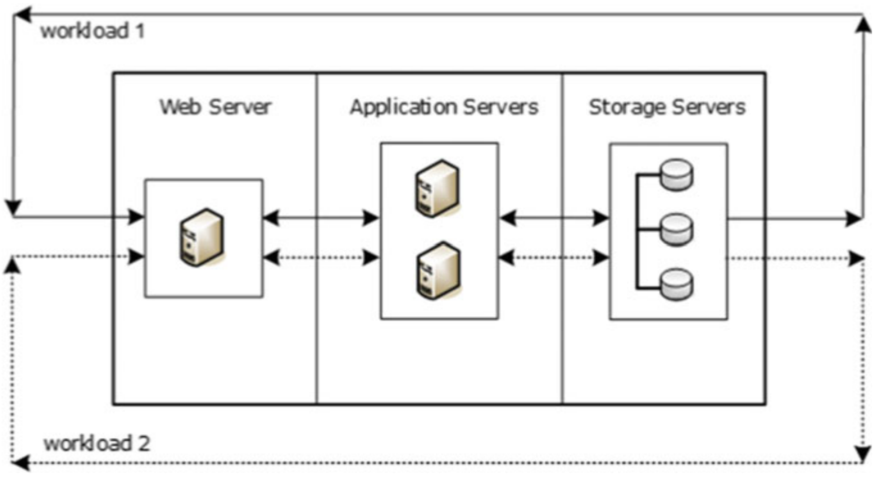


Fig. 3.6 Layout of the data center considered

The mean number of their requests that are in execution simultaneously is 100. According to forecasts, the number of employees authorized to access the servers is expected to double over the next nine months. The management is concerned that performance may degrade to an unacceptable level. Several initiatives are considered.

As a first action it is required to investigate the impact on performance of an increase of the global number N_0 of requests in execution from 100 to 200. More precisely, it is required to know if the QoS in terms of the mean per-class `Response times` defined for 100 employees are still satisfied. Since it is not possible to know how the fractions of the requests in execution of the two classes vary over time, to compute the upper bound of `Response times` it is necessary to consider all the possible combinations of the two classes of requests in execution, i.e., all the *population mix*, with $N_0 = 200$ req.

The capacity planning study should answer to several questions such as "Does the data center with the current configuration support the increase of the workload without saturating one or more resources?", "With $N_0 = 200$ *req* in execution will the QoS targets on the `per-class Response times` be satisfied?", "Which is the resource that is the bottleneck of the system?".

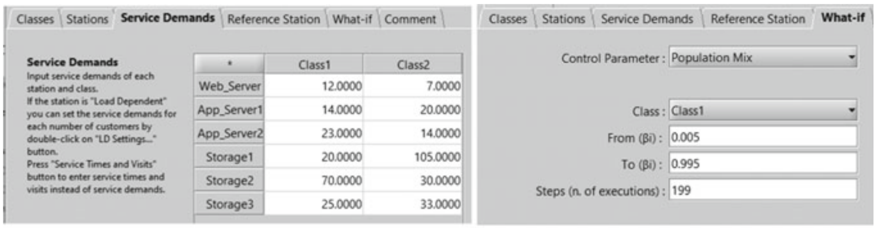
Other important questions to be answered concern the actions that should be taken to increase, if possible, the performance of the data center with the *current* configuration: "Which is the impact of the population mix on the potential increase of performance?", "The utilization of the resources are balanced?", "Which is the population mix that *maximizes* the `System Throughput` and *minimize* the mean `System Response time` (referred to as *optimal population mix*)?".

These questions have been grouped into the two *Objectives* of the study that are analyzed below.

3.3.2 Model Implementation

We need to evaluate the performance of the data center with an overall number of customers doubled (*200*) with respect to the current one (*100*). We implement a closed queueing model with six queue stations (see Fig. 3.6) and $N_0 = 200$ req in execution. The workload $\mathbf{N_0} = \{N_{0,1}, N_{0,2}\}$ consists of two classes of requests, where $N_{0,1}$ and $N_{0,2}$ are the number of requests in execution of $class1$ and $class2$, respectively.

To characterize the two type of applications in terms of processing requirements, a set of `Service demands`, one for each resource and each class, is used. The `Service demand` $D_{r,c}$ of a request of class-c at resource r is the total amount of time the request requires at that resource in order to be completely executed. The $D_{r,c}$ are computed *ignoring contention* by other requests and may be estimated measuring utilizations and throughputs and using the equation $U_{r,c} = X_{0,c}\, D_{r,c}$, where $X_{0,c}$ is the `system throughput` for class-c requests and $U_{r,c}$ is their utilization of resource r. To minimize the errors in the *parameter estimation*, it is *recommended*, when possible, to collect the measurements when the two types of appli-



Service Demands
Input service demands of each station and class.
If the station is "Load Dependent" you can set the service demands for each number of customers by double-click on "LD Settings..." button.
Press "Service Times and Visits" button to enter service times and visits instead of service demands.

*	Class1	Class2
Web_Server	12.0000	7.0000
App_Server1	14.0000	20.0000
App_Server2	23.0000	14.0000
Storage1	20.0000	105.0000
Storage2	70.0000	30.0000
Storage3	25.0000	33.0000

Control Parameter :	Population Mix
Class :	Class1
From (βi) :	0.005
To (βi) :	0.995
Steps (n. of executions) :	199

(a) `Service Demands` of the original workload (b) All mixes with the two classes

Fig. 3.7 `Service demands` of the two classes of requests (**a**), and `What-if` parameters (**b**)

cations are executed in isolation. The `Service demands` of the two classes of requests, in *ms*, are shown in Fig. 3.7a. The amount of work requested from the `Web Server` is much less demanding than the one requested from the `Application` and `Storage Servers`. The computations required by the business logic place a medium load on the `Application Servers` while the high number of data manipulated, uploaded and downloaded, generate a high load on the `Storage Servers`.

The `Service demands` are exponentially distributed and the scheduling discipline of the servers is processor sharing PS. These assumptions allows us to solve the model analytically with the MVA algorithm [25, 31] using the JMVA. Indeed, according to the BCMP theorem (see Sect. 3.1) , multiclass models in which the queue stations adopt the PS scheduling discipline can be solved analytically with efficient algorithms also if the service times (as may be considered the service demands of the single visit used our case) of the two classes at the same resource are *different*. Models in which these assumptions are not satisfied (e.g., if FCFS scheduling is required), must be solved with the simulation technique using JSIM.

The performance predictions obtained by the capacity planning study are based on several `What-if` analyses.

To evaluate the performance metrics corresponding to all the possible population mix we used the `What-if` feature of JMVA varying from 100% to 0% the requests in execution of one class and the opposite (from 0% to 100%) the fraction of the other class.

3.3.3 Results

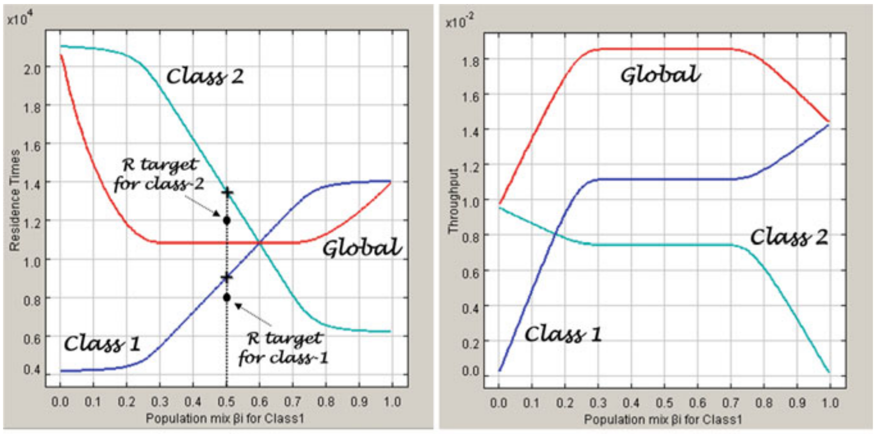
The peculiarity of the *workload forecasting strategy* adopted in this study is that *only* the *global intensity*, i.e., the total number $N_0 = 200$ requests in execution is known, and it is *not* possible to predict how the fractions of the requests of the two classes (i.e., the *population mix*) *vary over time*. Typically, it may be quite bursty.

Thus, to achieve the capacity planning goals `What-if` analyses are required with `population mix` as control parameter, Fig. 3.7b. The fraction β_1 of class-1 requests in execution range from 0.5% to 99.5%, the one of class-2 is the complement to 100%.

Obj.1: Evaluate the behavior of the performance indexes of the data center with a global `Number of requests` in execution $N_0 = 200$ and for all the possible combinations of the two classes.

JMVA is used to estimate the performance of the data center for the required parameter range with the `Service demands` shown in Fig. 3.7a. In the `What-if` screenshot of Fig. 3.7b the control parameter is `population mix`, and 199 models are executed with *all* the possible mix of the requests of the two classes ranging from $\mathbf{N_0} = \{1, 199\}$ to $\mathbf{N_0} = \{199, 1\}$.

Let's start with the analysis of the behavior of `per-class` and `Global System Response times` with respect to all the population mix with $N_0 =$ 200 req shown in Fig. 3.8a. The x-axis represents the fraction of class-1 requests β_1 with respect to the total number of requests in execution. In the two extremes $\beta_1 = 0$ and $\beta_1 = 1$ the workload consists of a *single* class, class-2 and class-1 only, respectively. In these cases, the resource that limit the performance of the system, i.e, the bottleneck, corresponds to the one with the max service demand D_{max}. When the bottleneck is saturated ($U_{bott} = 1$), the values of `System Response times` can be easily computed considering the D_{max} and applying the *Utilization* and *Little* laws, see Sect. 1.2. With **only** class-2 requests ($\beta_1 = 0$), the bottleneck is `Storage1` with $D_{Sto1} = D_{max} = 0.105$ s, from the *Utilization* law $U_{Sto1} = X_0 D_{max} = 1$ we derive



(a) `System Response Times` (b) `System Throughputs`

Fig. 3.8 `System Response times` [ms] (**a**) and `System Throughputs` [*req/s*] (**b**) with $N_0 = 200$ req *versus* fraction of class-1 requests in execution, from 0.5% to 99.5%

$X_0 = 1/D_{max}$ and by *Little* law it will be $R_0^{max} = N_0 D_{max} = 200 \times 0.105 =$ **21 s**. In the other extreme $\beta_1 = 1$ with **only** class-1 requests, the bottleneck is `Storage2` with $D_{Sto2} = D_{max} = 0.070$ s and it will be $R_0^{max} = N_0 D_{max} = 200 \times 0.070 =$ **14 s**.

As soon as the number of class-1 requests in execution increases (and thus class-2 requests decrease), the load of `Storage2` starts to grow. As a consequence, the bottleneck tends to migrate from `Storage1`, i.e., the class-2 bottleneck, to `Storage2`, i.e., the class-1 bottleneck. The corresponding class-1 `System Response time` $R_{0,1}$ increases until 13.86 s with the workload $\mathbf{N_0} = \{199, 1\}$, very close to the asymptotic value 14 s above computed. It does not coincide with it because with 199 req of class-1 and 1 req of class-2 the `Utilization` $U_{Sto2,1}$ of `Storage2` for *class-1 req.* is 0.995 and not 1. Similar motivations apply with the opposite workload $\mathbf{N_0} = \{1, 199\}$, i.e., $\beta_1 = 0.005$, where it is $U_{Sto1,2} = 0.995$ and the class-2 `System Response time` $R_{0,2}$ is 20.85 s while the asymptotic value is 21 s.

As can be seen in Fig. 3.8a, the `Global System Response time` R_0 is practically constant for a wide range of mixes, approximately between 30% and 70% of class-1. The important feature of these mixes is that executing a workload with one of them will result in the saturation of two resources *simultaneously*.

This interval of joint saturation, referred to as *common saturation sector*, is important in order to find the load of the system that optimize the performance, i.e., `Throughput` maximization and `Response time` minimization.

The identification of this interval can be done analytically under the assumption that the workload in execution is very large so that the bottleneck(s) is saturated [3]. With our workload of *200* req. the extremes of this interval are only approximate since the load does not saturate completely the bottlenecks.

The `Response times` of the two classes are identical when the two bottlenecks are *equiloaded* (for $\beta_1 = 0.6$ it is $R_0, 1 = R_0, 2 = 10.8$ s) and it can be shown that the corresponding *equiload* mix lies *inside* the common saturation sector [35]. R_0 is *minimum* in correspondence to this mix.

Figure 3.8b shows the `System Throughput`, `Global` X_0 and `per-class` $X_{0,1}$ $X_{0,2}$. It is evident that X_0 is *maximized* for all the mix of requests that belong to the common saturation sector while the per-class throughput are *constant* in the interval. It can be shown that the *equiutilization* point, i.e., the mix of the two classes that causes two bottlenecks to be equally utilized, lies into this interval and provides the *optimal load* that *maximizes* [35] the `global System Throughput`.

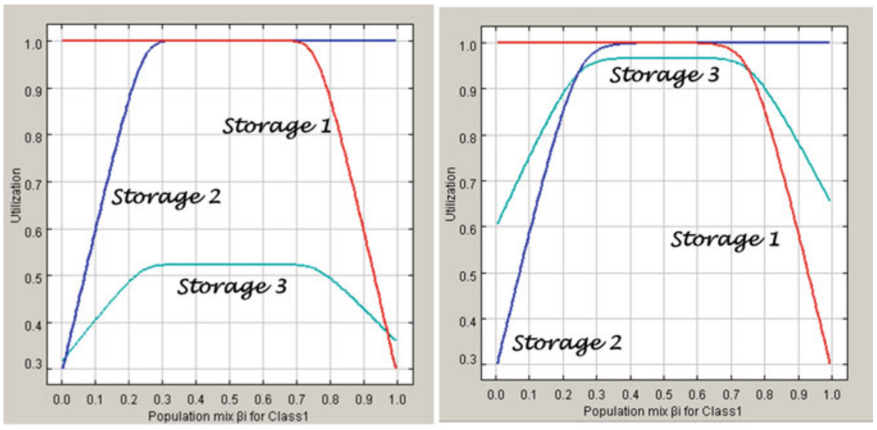
The behavior of `Response times` and `Throughputs` in Fig. 3.8 can be understood by analyzing the bottleneck *migration*. Indeed, it is known that the resource that limit the performance of the system under all possible workload mix is the bottleneck. So, when the bottleneck changes also the performance changes.

The `Utilizations` of the three `Storage Servers` of Fig. 3.9a show graphically this phenomenon. We do not consider `Web` and `App` servers because their `Service demands`, see Fig. 3.7a, are definitively lower than those of `Storage` servers, and therefore will never be saturated. As predicted, `Storage1` and `Storage2` saturate *together* for all the mixes of the common saturation sector (approximately between 30% and 70% fractions of class-1 *req.* of the total population of 200 req.) while the `Utilization` of `Storage3` is definitively lower (its max `Utilization` reached in the common saturation sector is $U_{Sto3} = 0.52$), since its `Service demands` are smaller with respect to those of `Storage1` and `Storage2`. When only a few class-1 requests are in execution ($\beta_1 < 0.3$), the bottleneck is `Storage1`. On the other side, when the fraction of class-1 req in execution is high ($\beta_1 > 0.8$) *only* `Storage2` is saturated.

With regard to the *Quality of Service* targets on the `per-class System Response times` that were set for the original configuration of the data center with a workload of $N_0 = 100$ req evenly divided between the two classes, i.e., $\boldsymbol{\beta} = \{0.5, 0.5\}$, we can see that with $N_0 = 200$ req *cannot be satisfied*. Indeed, the *target* values assigned to the mean `System Response times` of the two classes were $R_{0,1} = 8$ s and $R_{0,2} = 12$ s, respectively. With $N_0 = 100$ req these values **were met** ($R_{0,1} = 4.5$ s and $R_{0,2} = 6.7$ s), with $N_0 = 200$ req otherwise they **are not** ($R_{0,1} = 9$ s and $R_{0,2} = 13.5$ s), see Fig. 3.8a. We will see in the following *Obj.2* that a load balancing action allows the satisfaction of the targets.

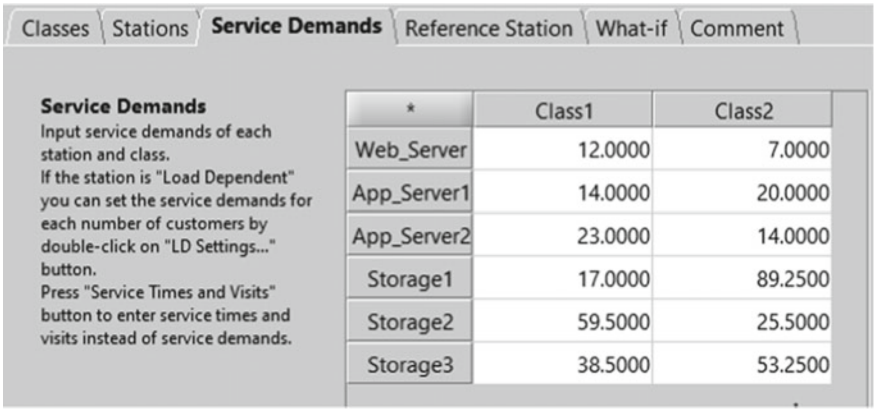
Obj.2: Investigate on the actions that may improve the performance of the data center with the current configuration and a workload of N_0 = 200 req. Can the `System Response time` objectives be achieved after these actions?
Figure 3.9a emphasizes the problem: the unbalanced utilization of the `Storage Servers`.

To enhance the performance we need to take into consideration the most heavily loaded servers, namely `Storage1` and `Storage2`, since reducing the `Service demands` at resources other than the *bottlenecks* produces only *marginal* improvements. The *total load* on the two `Storage Servers` 1 and 2 is fairly balanced, $D_{Sto1} = 125$ ms and $D_{Sto2} = 100$ ms, while the load of the third server is much



(a) `Utilizations` with *original* workload (b) `Utilizations` with *balanced* workload

Fig. 3.9 `Utilizations` of the `Storage servers` in the *original* configuration (**a**) and in the *balanced* system (**b**) *vs* population mixes. The workload ranges from {0,200} to {200,0} req



Service Demands
Input service demands of each station and class.
If the station is "Load Dependent" you can set the service demands for each number of customers by double-click on "LD Settings..." button.
Press "Service Times and Visits" button to enter service times and visits instead of service demands.

*	Class1	Class2
Web_Server	12.0000	7.0000
App_Server1	14.0000	20.0000
App_Server2	23.0000	14.0000
Storage1	17.0000	89.2500
Storage2	59.5000	25.5000
Storage3	38.5000	53.2500

Fig. 3.10 `Service demands` for the *balanced* configuration of the three `Storage Servers`

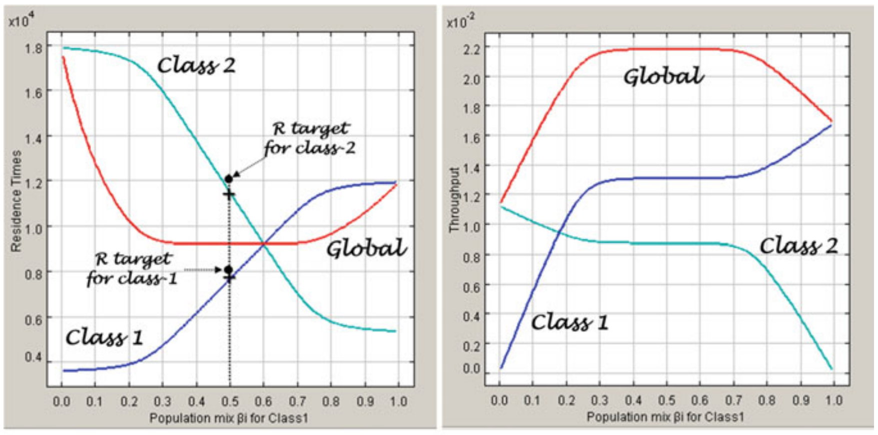
smaller, 58 ms, see Fig. 3.7a. We want to assess the effect of alleviating the bottlenecks, trying to balance the loads of all three `Storage Servers`. So, even according to usage statistics, some files have been migrated between the various storage, more precisely from `Storage1` and `Storage2` to `Storage3`, in order to make their total `Service demands` more similar to each other than in the original configuration. Figure 3.10 shows the new `Service demands`.

Let us remark that the `Global Service demand` to all the three `Storage Servers` must remain the same as the one of the original configuration, namely 283 ms, since all the servers have the same technical characteristics. Thus, shifting some data from one resource to another alter the visits V_r but not the `Service time` S_r of a request.

This configuration denotes a good balancing of the load of the three storages and no server is underutilized, Fig. 3.9b. By comparing their behavior with those of Fig. 3.9a obtained with the original system it is evident that the sum of the three U_r is *maximized*, thus enabling the maximization of the `System Throughput`.

The *maximum* `System Throughput` of the *original* system, obtained with fractions inside the common saturation sector, was $X_0^{max} = 0.0185$ req/ms, Fig. 3.8b, while the one obtained after the *balancing* action, Fig. 3.11b, is $X_0^{max} = 0.0218$ req/ms, with an *improvement* of about **17.8%**.

The corresponding *minimum* `System Response time` were $R_0^{min} = 10.8$ s for the *original* system, Fig. 3.8a, and $R_0^{min} = 9.18$ s for the *balanced*, Fig. 3.11a, with a *reduction* of about **15%**.



(a) `System Response Times` (b) `System Throughputs`

Fig. 3.11 `System Response times` [*ms*] and `Throughputs` [*req/ms*] in the *balanced* configuration *versus* population mixes. The workload ranges from {0,200} to {200,0} req

With regard to the targets defined for the `per-class System Response times`, i.e., $R_{0,1} = 8$ s and $R_{0,2} = 12$ s, with the mix $\boldsymbol{\beta} = \{0.5, 0.5\}$, we can see from Fig. 3.11b that the *balanced* configuration is **able to satisfy** them, i.e., $R_{0,1} = 7.65$ s and $R_{0,2} = 11.47$ s. Instead, as described previously, the original configuration of data center with 200 req in execution was **unable** to reach them.