Service Engineering
May 2000

An Introduction to Skills-Based Routing and its Operational Complexities

By Ofer Garnett and Avishai Mandelbaum
Technion, ISRAEL

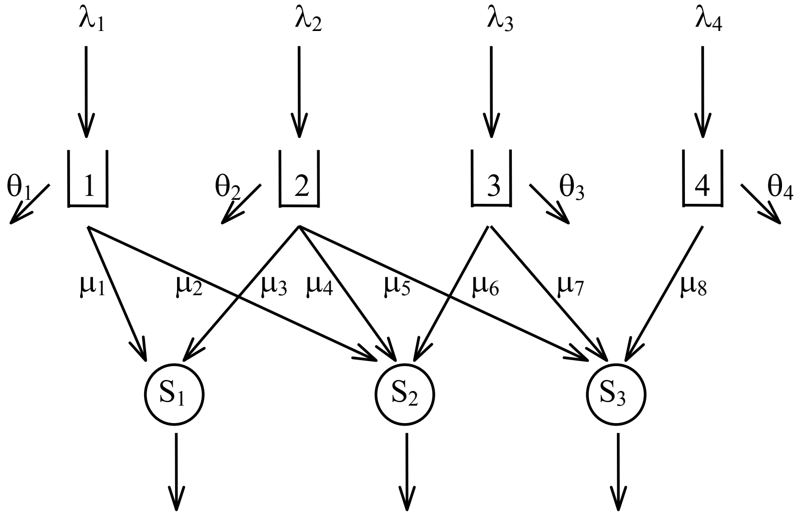
(Full Version)

Contents:

Acknowledgement: This teaching-note was written with the financial support of the Fraunhofer IAO Institute in Stuttgart, Germany. The authors are grateful to Dr. Thomas Meiren and Prof. Klaus-Peter Fähnrich of the IAO for their assistance and encouragement.

Introduction

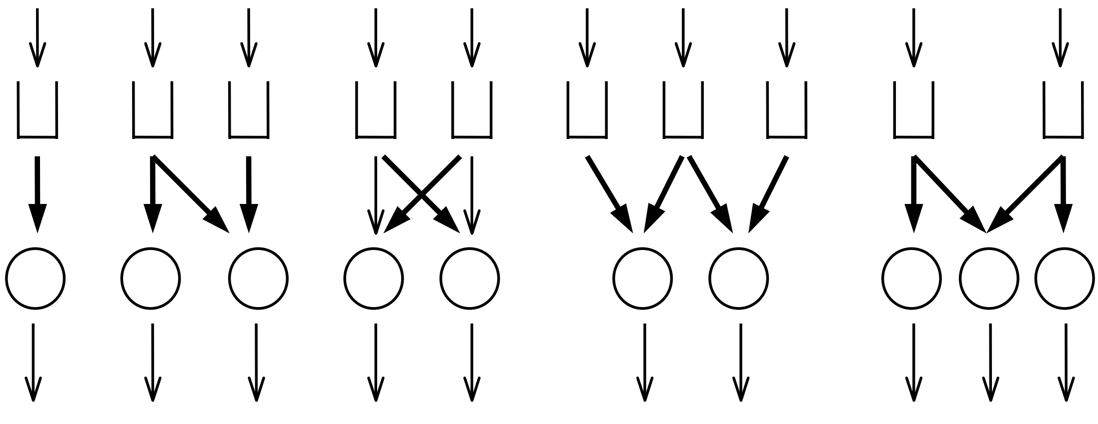
Consider the following multi-queue parallel-server system (animated, for example, by a telephone call-center):



Here the λ's designate arrival rates, the μ's service rates, the θ's abandonment rates, and the S's are the number of servers in each server-pool.

Such a design is frequently referred to as a **Skills-Based** design since each queue represents "customers" requiring a specific type of "service", and each server-pool has certain "skills" defining the services it can perform. In the diagram above, the arrows leading into a given server-pool define its skills. (For example, a server from pool 2 can serve customers of type 3 at the of rate μ_6 customers per unit of time) .

Some canonical designs are: I (I^k), N, X, W, M (V).



When implementing a skills-based system, the major decisions to be made are:

1. Who are the customers - defining customer *types* (*offline* decisions)
2. Who are the servers - their *skills* and numbers (*offline*)
3. How are customers *routed* to servers - the *control policy* (*online*)

These decisions typically require the involvement of separate divisions in the organization: Marketing (for no. 1), HRM (for 2) and Operations (2, 3), all supported by MIS / IT – a truly multi-disciplinary challenge.

System's *design* (engineering) consists of classifying the customers and determining the servers' required skills. A particular single design can have alternative interpretations, for example:

- Different customer types can represent customers requiring different services (e.g. technical support vs. billing) or customer priorities (VIP vs. Members).
- Separate server-pools can be due to servers' level of capabilities / training / experience (e.g. Hebrew/English speaker vs. Arabic/Russian/Spanish speaker, generalist vs. specialist, expert vs. novice).

Even with simple designs there can be associated many different *control* (routing) *policies*. The two most common decisions to be made when routing customers are:

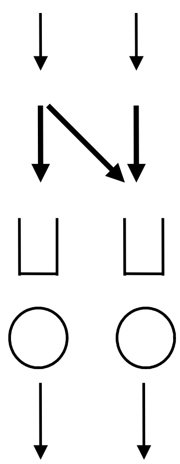
1. Whenever a service ends and there are queued customers, which customer (if any) should be routed to the server just freed.
2. Whenever a customer arrives and there are idle servers, to which one of them (if any) should the customer be routed.

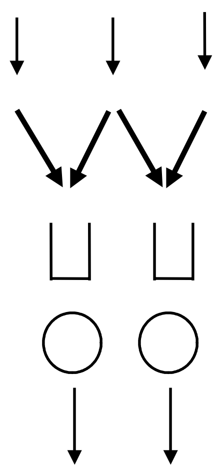
Skills-Based Routing (SBR) is the protocol for online routing of customers. Routing decisions can be *dynamic* - depending on the "state" of the system at the time they are made, or *static* – for example, each server pool adheres to static priorities among its constituent types.

The design of SBR protocols presents challenging research problems, and the goal of our teaching note is to contribute to their understanding. We achieve this through performance analysis of alternative system's designs and staffing levels. The analysis is predominantly *simulation-based* since even for relatively simple designs, such as V- or N-design, the available analytic state-of-art is rather restricted. In our simulations we are using Poisson arrivals, as models of completely random arrivals, and exponential service times, which quantifies human stochastic variability. Customers' patience (in models including abandonment) is assumed exponential - although not backed up empirically, this serves as a prevalent basic Markovian approximation. (Details on the simulations are given in the Technical Appendix.)

Note: a design with a multi-server pool is mathematically equivalent to a design in which the pool is replaced by multiple (as many as in the pool) equally-skilled-single-servers.

Note: Sometimes one must associate the queues with the server pools rather than with customer types. A control policy then amounts to routing customers upon their arrivals, as depicted in the following N- and W-designs.

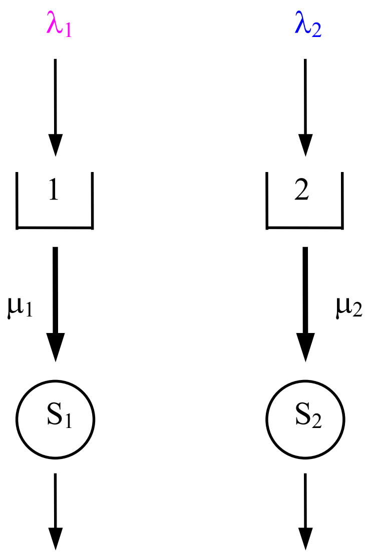




N-design with single servers (no abandonment)

Consider a help-desk with two servers providing technical support for two products. The current design is of type I^2 (see diagram below) - each server "specializes" in one product, and serves only the customers requiring technical support for that product. **Type 1** are core customers – they are considered higher priority than **type 2** and hence served by an expert **server 1**.

We analyze this system analytically as two independent M/M/1 queues.



The system's parameters are:

$\lambda_1 = 1.3\rho$, $\lambda_2 = 0.4\rho$,

$\mu_1 = \mu_2 = 1.0$,

$S_1 = S_2 = 1$,

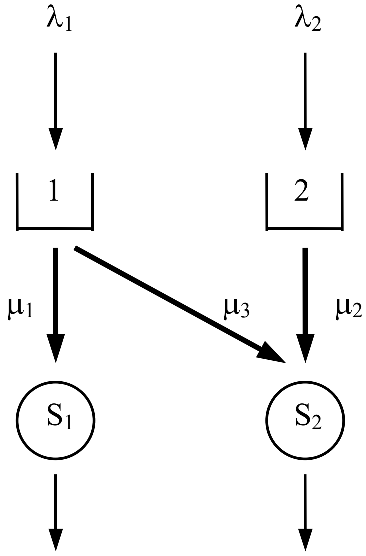
where ρ parametrizes the traffic load, and the given rates are per minute.

When the traffic load is low the performance is satisfactory (see Table 1 below), but as traffic increases - specifically as ρ approaches 1/1.3 - **server 1** becomes heavily loaded and **type 1** customers suffer long waiting times. This situation raises questions as to this design's efficiency, since **server 2** is still fairly "comfortable" with the higher traffic load, and frequently goes idle while **server 1** is faced with a long queue. Perhaps more importantly, with the higher traffic load, the service-level of the high priority **type 1** customers deteriorates dramatically.

Table 1 : Performance of I^2 design						
	Type 1 Customers / Server 1			Type 2 Customers / Server 2		
Traffic Load (ρ)	Average Wait (seconds)	Average Queue Length	Utilization	Average Wait (seconds)	Average Queue Length	Utilization
1/3	46	0.3	43%	9	0.0	13%
2/3	390	5.6	87%	22	0.1	27%
3/4	1170	38.0	97.5%	26	0.1	30%

Can alternative designs be more efficient and provide better service? The answer is an emphatic "Yes", as will now be unraveled through a sequence of such designs.

The first alternative is to implement a V-design(1) with a pool of two servers. This is clearly the most efficient design utilization-wise. However, it requires that the expert **server 1** caters to customers of **type 2**, which is judged unreasonable.



$\lambda_1 = 1.3\rho \qquad \lambda_2 = 0.4\rho$

$\mu_1 = \mu_2 = 1.0 \qquad \mu_3 = 0.5$

$S_1 = S_2 = 1$

Another possibility is an N-design(2) (see diagram above): **server 2**, who is not as busy as **server 1**, could *help* by serving some of the **type 1** customers. This requires additional training for **server 2** - to be able to support **type 1** customers although at a slower rate ($\mu_3 = 0.5$).

Two *control policies* will now be considered, which are referred to as "greedy" in the sense that **server 2** does not remain idle when there are customers (of any type) waiting in queue. The policies differ by whether **server 2** gives priority to **type 1** or **type 2** customers:

Greedy 1: **Server 2** serves **type 1** customers as long as there are **type 1** customers waiting and **server 1** is busy.

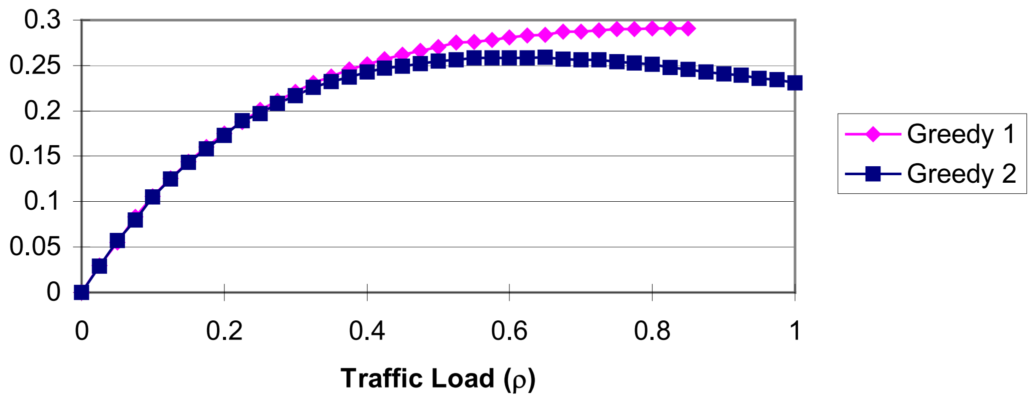
Greedy 2: **Server 2** serves **type 2** customers as long as there are **type 2** customers waiting, and otherwise serves any **type 1** customers waiting if **server 1** is busy.

Implicit in the description is that when both servers are idle then **type i** customers are routed to **server i,** i=1,2.

The N-design was implemented so that **server 2** could "assist" **server 1** in serving the high priority **type 1** customers. As will be explained below, the N-design is indeed more efficient than I^2 : it can handle in principle any $\rho \leq 1$ workload while I^2 is restricted to $\rho \leq 1/1.3 < 1$.

Question: How much "assistance" does **server 1** get from **server 2** in each case ?

Graph 1 : Fraction of Type 1 served by Server 2



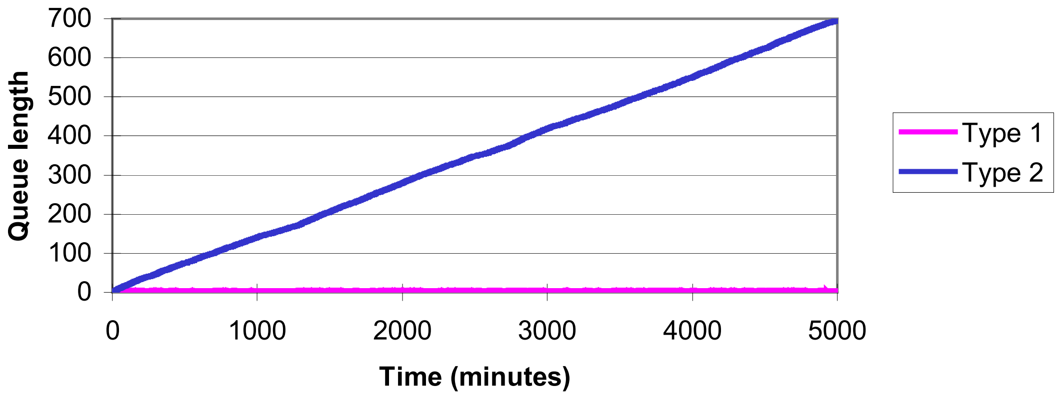
When the traffic load is low there is little assistance (with similar values for both policies) since **server 1** easily handles the flow of **type 1** customers, and a queue rarely forms. When $\rho \cong 1/3$ we start to see the anticipated difference between the two policies - more assistance is given under "Greedy 1", and the higher the traffic load the more assistance is given. Under "Greedy 2" with traffic loads above a certain point ($\rho \cong 3/5$) the amount of assistance given actually starts to decrease. In this regime **server 2** cannot assist as much, being busy with **type 2** customers that enjoy higher priority.

It is significant that the graphs of both policies do not "end" at the same value of ρ. This will be discussed momentarily.

For a given set of parameters and a *control policy*, the system can be either *stable* or *unstable*. Consider the following stability heuristics for the case ρ = 1: If **server 1** devotes all his time to serving **type 1** customers, **server 2** will have to serve an average of $\lambda_1 - \mu_1 = 1.3 - 1.0 = 0.3$ **type 1** customers per minute, which requires 0.3 / $\mu_3 = 0.3/0.5 = 60\%$ of his time. The remaining 40% of **server 2** time are just adequate to serve the 0.4 / $\mu_2 = 0.4/1.0$ workload of **type 2** customers.
(Recall that the original I^2-type design became unstable for $\rho \geq 1.3$).

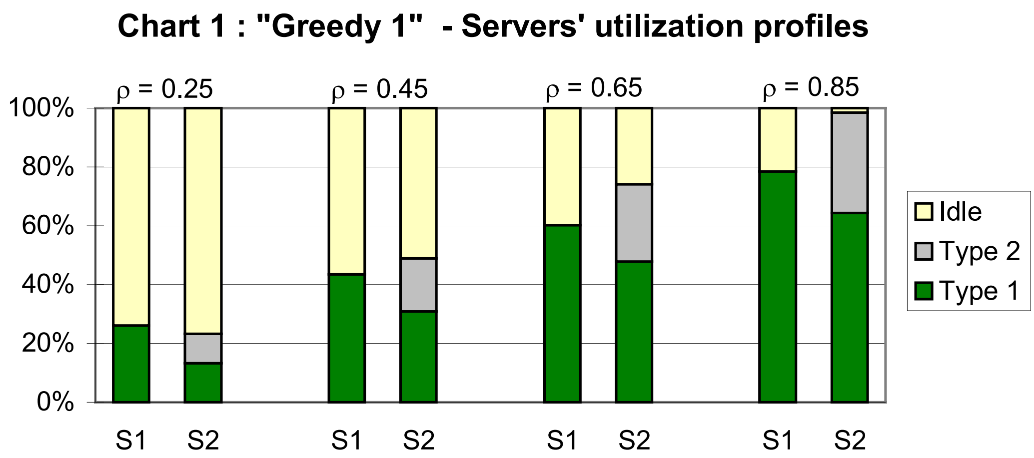
Question: Does this mean that the system will be *stable* under both "greedy" *control policies* for any $0 < \rho < 1$?

Graph 2 : "Greedy 1" , ρ = 0.95 - Queue lengths



The graph displays a constant linear growth of the queue-length for **type 2** customers under the "Greedy 1" policy - clearly indicating instability. (The queue of **type 1** is negligibly flat relatively to **type 2**.)
Returning to Graph 1: The graph for the "Greedy 1" policy does not reach ρ = 1 since the system becomes unstable at $\rho \cong 0.9$. Under "Greedy 2", ρ = 1 is reached, and indeed the fraction of **type 1** customers served by **server 2** approaches the expected value of (1.3 – 1)/1.3, somewhat below 0.25.

Question: Can you explain what caused the system, under the "Greedy 1" policy, to become unstable before reaching $\rho = 1$, and why the stability analysis preformed does not apply to this case ?



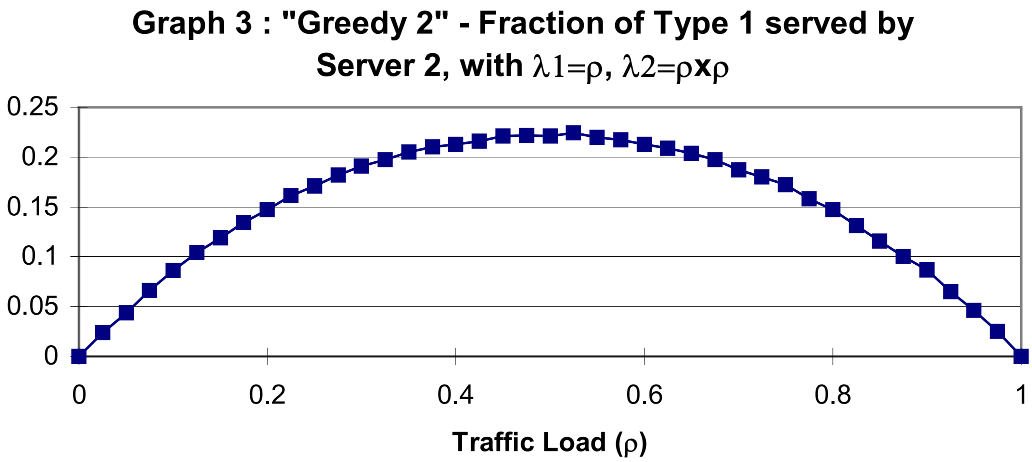
Note the servers' utilization profiles (Chart 1 above) for the case $\rho = 0.85$: **Server 2** is almost constantly busy, while **server 1** is idle more than 20% of the time. Moreover, **server 2** spends most of his time serving **type 1** customers. From here it is clear that **server 2** is "assisting" beyond the call of duty – in fact, he is overworking himself by taking away work that **server 1** could and should have handled by himself. This is the cause for instability : **Server 2** gives so much attention to **type 1** customers that his own neglected queue "explodes".

It is important to realize that we are unable to determine beforehand the maximum workload such a "Greedy 1" system can handle. The stability analysis does not apply to this case since the underlying basic assumption that both servers are fully utilized does not prevail here.

Note: Already from Graph 2 follows that, with $\rho = 0.95$, **type 2** customers accumulate at a rate of 700/5000 = 0.14 per minute. The offered traffic of **type 2** is 0.4 x 0.95 = 0.38, hence **type 2** are served at a rate of 0.38 – 0.14 = 0.24. This translates into only 24% of **server 2** time, with the rest 76% devoted mostly to **type 1**.

For a more complete understanding of server "assistance", assume that the N-design was better "balanced" in that the "assistance" given by **server 1** was not vital for keeping **server 2** from being overloaded, for all values of $\rho < 1$. In other words, the arrival rates (λ_1 and λ_2) were such that the original I^2-type design was stable for all $\rho < 1$. An example of such a scenario is: $\lambda_1 = \rho$ and $\lambda_2 = \rho^2$. Unlike the original setup, where at $\rho = 1$ **server 1** is overloaded with **type 1** customers and must receive "assistance" from **server 2** (who has not reached full capacity serving **type 2** customers), here both servers just reach their full utilization by serving their own "types" (they are "balanced".).

Question: How do you expect the graph for the "Greedy 2" policy to behave as ρ approaches 1 in this case ?



This "completes" the picture (Graph 1) for the "Greedy 2" case - when the traffic load is very high (approaching $\rho = 1$) **server 2** is so busy with **type 2** customers that he is not able to assist **server 1**, whereas in the previous case (Graph 1) even as ρ approaches 1 **server 2** is not fully utilized serving just the **type 2** customers and therefore "assists" the overloaded **server 1** by serving part of the **type 1** customers.

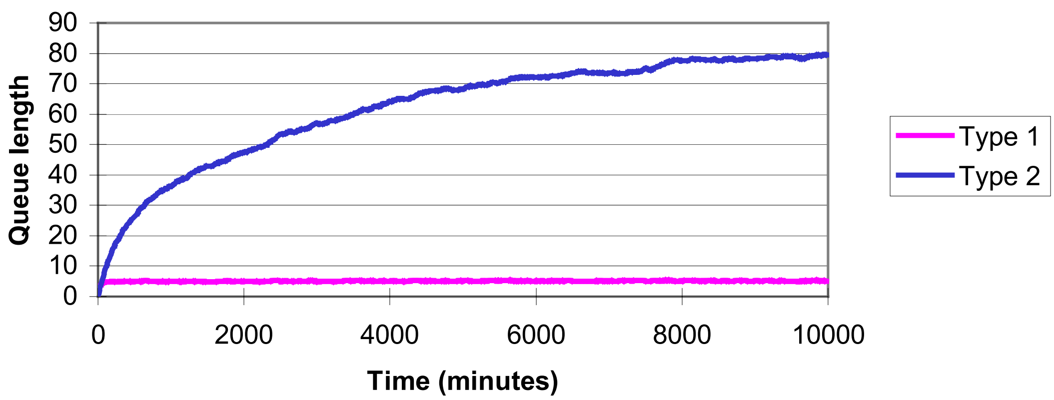
We have seen that the "Greedy 1" policy dictates an inefficient division of the workload, preventing the system from reaching its full processing potential (i.e. $\rho = 1$). Nonetheless this policy has the desirable feature that the higher-priority **type 1** customers indeed enjoy high priority, contrary to "Greedy 2". Therefore our discussion from here on will be based on the "Greedy 1" policy. ("Greedy 2" would have been more appropriate under the alternative, also realistic, scenario where **type 2** are the VIP's.)

To try and overcome the problems of the "Greedy 1" policy, we introduce the *single-threshold control policy*. This is a simple variation on the "Greedy 1" policy, with the following additional restriction: **server 2** assists **server 1** only when the queue of **type 1** customers is *at or above* a certain *threshold*.
Note that such a policy (except for the case of a threshold of 1 which is equivalent to the "Greedy 1" policy) is not *work conserving*. By this we mean that it is possible to have customers waiting in a queue while there is an idle server that is capable of serving them.

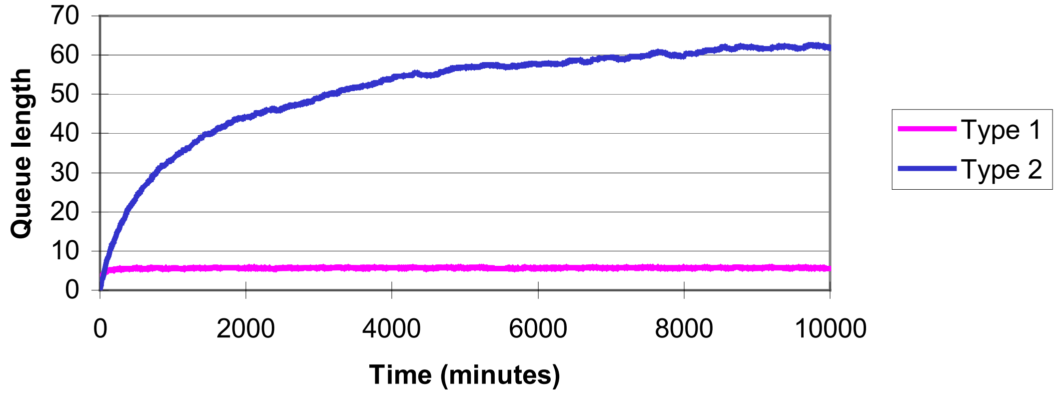
With a traffic load of ρ = 0.95, applying a single-threshold can lead to a stable system.

Question: Can you explain the effect of a threshold on the dynamics of the system?

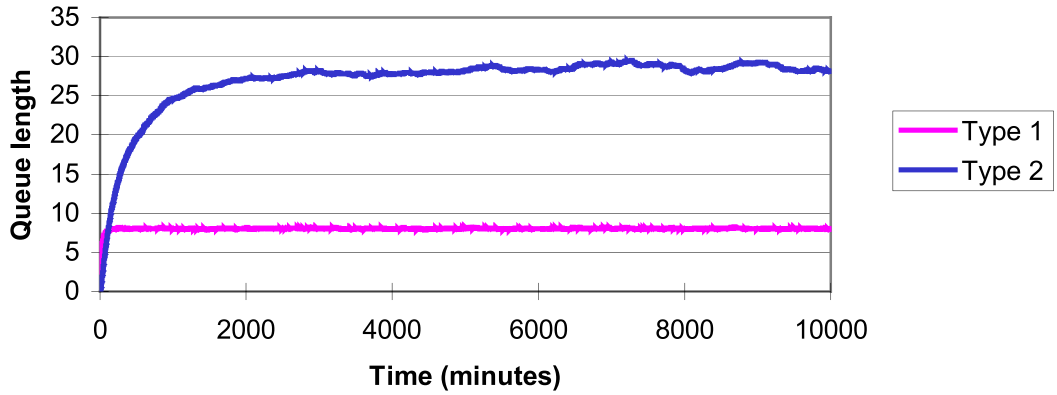
Graph 4 : Threshold = 4 , ρ = 0.95 - Queue Lengths



Graph 5 : Threshold = 6 , ρ = 0.95 - Queue Lengths



Graph 6 : Threshold = 8 , ρ = 0.95 - Queue Lengths



Comparing the graphs (4-6 above), one sees that the higher the threshold the longer the queue of **type 1** customers (its length is slightly above the threshold) and the shorter the queue of **type 2**; also the system reaches its steady-state faster (but this last trend reverses, as it turns out, for higher thresholds when the queue of **type 1** becomes dominant).

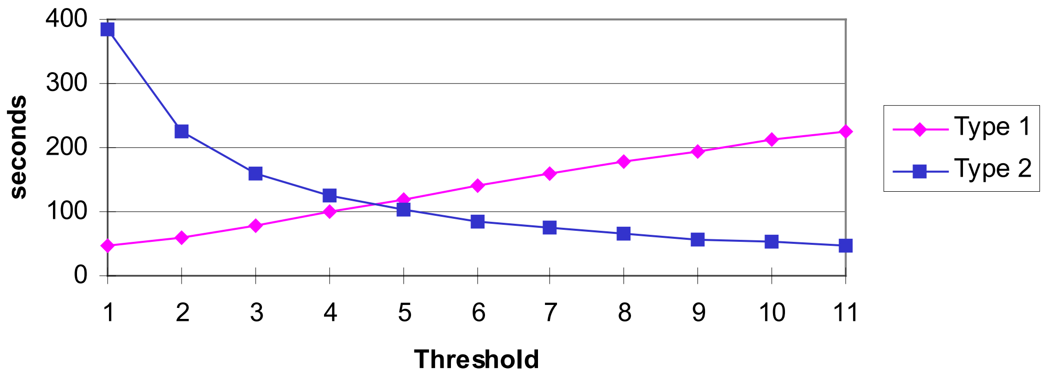
Having a threshold reduces the amount of "assistance" that **server 1** gets and prevents **server 2** from assisting when the workload on **server 1** is low (below the threshold). The threshold provides safety against such cases of "over-assistance" as described following Chart 1 above. With a threshold that is high enough, **server 2** should have enough time to serve the **type 2** queue, thus preventing the "explosion" exhibited in Graph 2. (In our case, such stability is achieved only for thresholds of 4 or greater.)

Note that a *single-threshold control policy* can, alternatively, be based on a *threshold* with respect to the queue of **type 2** customers, in which case **server 2** assists **server 1** only when the **type 2** queue is at or below the threshold. However, using such a threshold policy is not as effective as the one discussed so far – indeed, to serve as a successful "divider-of-labor", the threshold must relate to the queue which is served by both servers. This threshold of **type 2** lacks the essential characteristic of preventing **server 2** from needlessly assisting **server 1** when the **type 1** workload is low.

It is clear that the higher the threshold, the less "assistance" **server 1** receives.

Question: How does the threshold affect the average wait in queue of each customer type ?

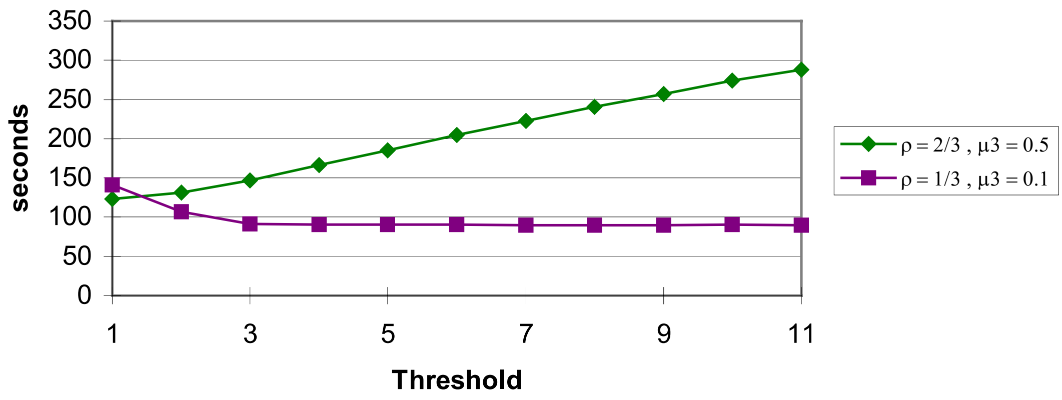
Graph 7 : ρ = 2/3 - Average Wait in Queue



Increasing the threshold allows for more cases in which **server 2** gives priority to **type 2** customers. As expected, this results in a decrease of the **type 2** customers' waiting times, and an increase of the **type 1** waiting times.

Question: How about the average total time that **type 1** customers spend in the system (i.e. wait + service, for **type 1** customers) ?

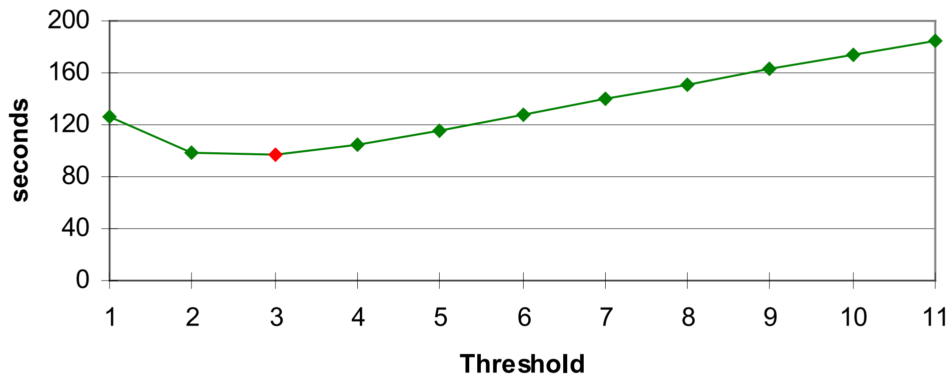
Graph 8 : Type 1 - Average Time in System



With the current parameters (ρ = 2/3, μ_3 = 0.5) the results are those intuitively expected - the **type 1** customers' average time in the system increases, similar to their average wait in queue. However, this is not true in general - changing the parameters (ρ = 1/3, μ_3 = 0.1) we get a <u>*"slow server"*</u> *effect*. Since **server 2** is so much slower serving the **type 1** customers (10 times slower than **server 1**) the **type 1** customers he serves spend a long time in the system. As in this example, the "damage" of the slow service is greater than the overall "benefit" for **type 1** customers, of reduced <u>waiting</u> times from having **server 2** assist **server 1**.

Note the change in the <u>overall</u> average wait as the threshold increases, revealing an "optimal" threshold (marked in Graph 9 below).

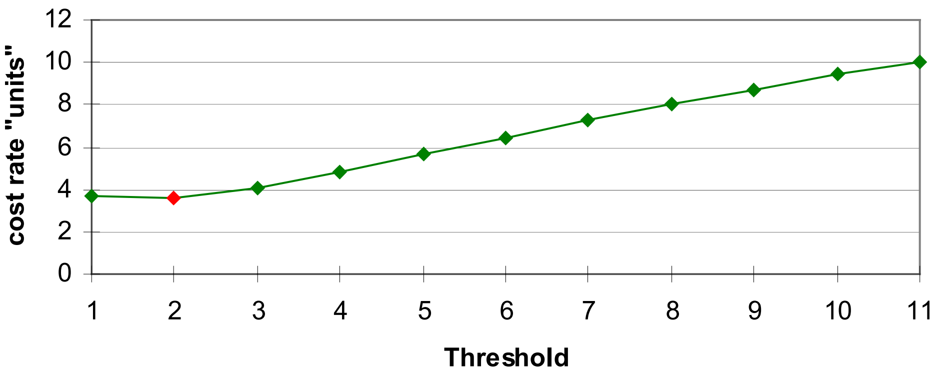
Graph 9 : ρ = 2/3 - Overall Average Wait in Queue



Recall that customers' priorities can be translated into "costs". Assume, for example, that the cost of a waiting customer is (linearly) proportional to his total wait in queue, and that **type 1** customers (high priority) accumulate cost at a rate three times higher than that of **type 2** customers. It is now possible to find an "optimal" threshold in the sense that the overall cost rate is minimal.

Question: Will this "optimal-cost" threshold necessarily be the same as the "optimal-wait" threshold found before (Graph 9 above) ?

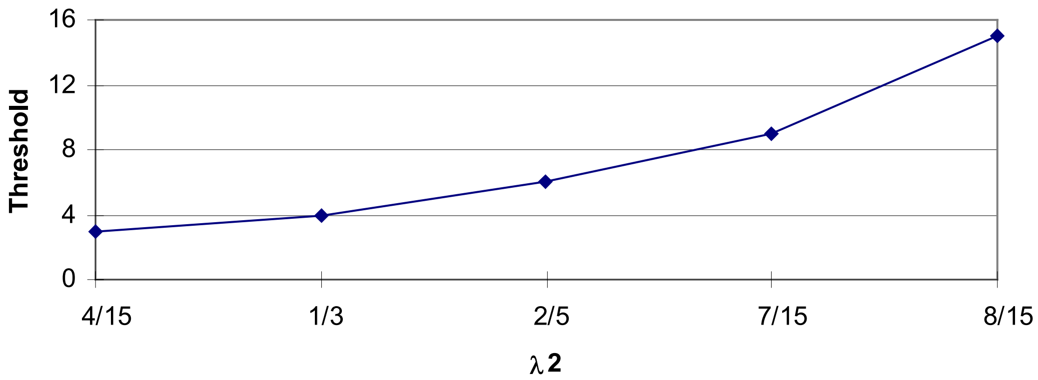
Graph 10 : ρ = 2/3 - Overall Cost Rate



Graph 10 displays the overall cost rate in cost rate "units" - 1 "unit" being the cost rate of a single **type 2** customer (a **type 1** customer has a cost rate of 3 units). Obviously the "optimal-cost" threshold is not identical to the "optimal-wait" threshold. Thus different methods for measuring performance of a system may lead to different "optimal" thresholds.

Question: Is the "optimal-wait" threshold sensitive to the traffic load of **type 2** customers ?

Graph 11 : λ1 = 1.3 x 2/3 - "Optimal" Thresholds



Graph 9 corresponds to $\lambda_2 = 0.4 \times 2/3 = 4/15$, under which the optimal threshold is 3. As the traffic load of **type 2** customers increases, so does the "optimal" threshold. This is consistent with the interpretation of the "optimal" threshold as a balance point between "starving" **server 1** - leaving him without enough work to keep him busy, and "neglecting" the **type 2** customers. Therefore with a higher **type 2** traffic load, a higher threshold is needed to retain the balance and prevent "neglect" of **type 2** customers.

Question: Reflecting on Graph 11 above, what *multi-threshold control policy* can you suggest as a natural refinement of the single-threshold policy ?

Thresholds : $C_0 < C_1 < C_2 < C_3 < C_4 < \ldots$

The single-threshold control policy discussed above has a threshold with respect to the **type 1** queue. A multi--threshold control policy also takes into account the **type 2** queue. Even if the average **type 2** traffic load is constant, the **type 2** queue length at a given instant represents the current workload of **type 2** customers. Since the "optimal" threshold increases with the **type 2** traffic load (Graph 11 above) it is reasonable to have a series of **type 1**-queue-thresholds depending on the **type 2** queue, which increases with the queue length.

No calculations are carried out for multi-threshold controls since, as explained below[(4)], significant improvements over the single-threshold are not very likely.

Note that for an N-design with multi-server pools, a threshold policy can also have a *server reservation* aspect - there can be different thresholds (for routing a **type 1** customer to a **type 2** server) depending on the number of idle **type 2** servers.

V- and N-designs: research on performance analysis and control

(1) An asymptotically optimal control policy for the general V-design is identified by J. A. Van Mieghem in "Dynamic scheduling with convex delay costs: the generalized c-μ Rule", *Annals of Applied Probability*, **5**, 808-833, 1995. This policy is structured as follows: service within a customer type is FCFS; each type is assigned an index which is an increasing function of the time that its first-in-queue has been waiting; finally, the next customer to be served is the one with the highest index.

(2) The parameters for our base N-model and few of the above results were adapted from the paper by J. M. Harrison: "Heavy traffic analysis of a system with parallel servers: asymptotic optimality of discrete-review policies", *Annals of Applied Probability*, **8,** 822-848, 1998.

(3) Optimality of a multi-threshold policy has been verified for the N-design, numerically via Dynamic Programming, by Y. Newman and Y. Nov (IE&M project, supervised in 1999 by Prof. M. Pollatscheck and A.M.) To the best of our knowledge, an analytical proof is still lacking

(4) In a paper by S. L. Bell and R. J. Williams ("Dynamic Scheduling of a system with

two parallel servers in heavy traffic with complete resource pooling: asymptotic optimality of a continuous review policy", *Preprint*, 1-43, 1999), it is shown that a single-threshold policy, as in Graphs 4-6, is *asymptotically* optimal for the N-design. Their parametric regime is like ours, where **type 1** customers are of higher priority, and the help of **server 2** to **server 1** is necessary for stability. (This differs from the regime in Graph 3, as in H. J. Kushner and Y. N. Chen, "Optimal control of assignment of jobs to processors under heavy traffic", *Preprint*, 1-48, 1998.)
Bell and Williams analyze a model with single server-pools and no customers' abandonment (with, moreover, possible *preemption* in the midst of a **type 2** service in favor of a **type 1** threshold.) Such restrictions are unrealistic for telephone call centers, which are our prime motivators. We continue therefore with a non-preemptive model (X-design) that accommodates multi-server pools and customers' abandonment.

X-design with multi-server pools and abandonment

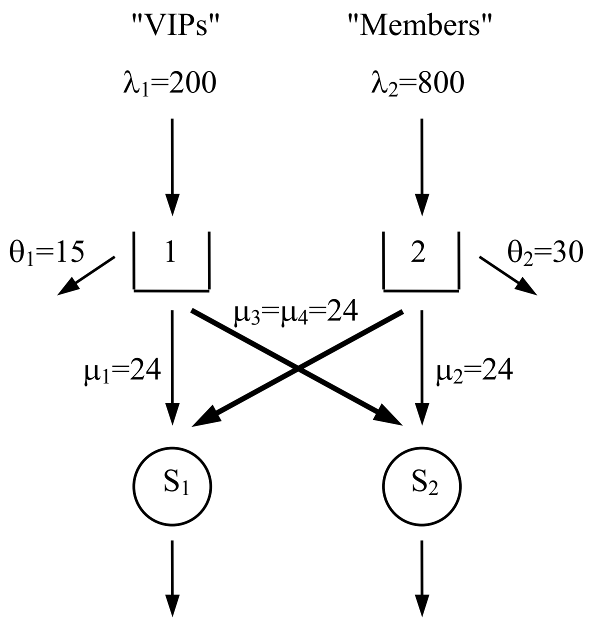
In the X-design each *server pool* can "assist" the other by serving the other pool's customers. Thus, all servers are "skilled" for all services, but may differ in levels of proficiency (resulting in different service rates).

Adding abandonment to the model eliminates the issue of stability: such a system is always stable – the heavier the traffic the heavier the abandonment, but the queues remain statistically stable.

Consider the following scenario : A call center providing customer service wishes to differentiate between high priority customers ("VIPs", constituting 20% of all calls) and low priority customers ("Members", 80% of calls). Assume that both types are characterized by the same service rate: $\mu_1 = \mu_2 = 24$ customers per hour, or an average of 2.5 minutes per call. Also assume that "VIPs" are more "patient" than "Members": an average "patience" of 4 minutes for "VIPs" and 2 minutes for "Members".

The service level at this call center is measured by the **ASA** (Average Speed of Answer = average time until call was answered, given it was answered). All servers are equally "skilled" to serve both types.

The X-design with its system's parameters is given as follows:



A number of operational experiments were performed, testing alternative designs and *control policies*. All policies tested were *work conserving* (<u>not</u> of the threshold-type demonstrated in the previous section). Three setups were selected as candidates (A, B and C described below). In all three cases the workforce totaled 35 servers, faced with a call volume of 1000 calls per hour.

<u>Setup A</u> : (X-design)

"VIP" servers : $S_1 = 20$

If "VIP" queue not empty serve the "VIP" queue + all "Members" waiting more than **40** seconds, as a single FIFO queue.

If "VIP" queue is empty, serve the first in the "Member" queue.

"Member" servers : $S_2 = 15$

If "Member" queue not empty serve the "Member" queue + all "VIPs" waiting more than **6** seconds, as a single FIFO queue.

If "Member" queue is empty, serve the first in the "VIP" queue.

<u>Setup B</u> : (X-design)

"VIP" servers : $S_1 = 10$

If "VIP" queue not empty serve the "VIP" queue + all "Members" waiting more than **120** seconds, as a single FIFO queue.

If "VIP" queue is empty, serve the first in the "Member" queue.

"Member" servers : $S_2 = 25$

Serve both queues as a single FIFO queue.

<u>Setup C</u> : (V-design; feasible since servers are assumed equally skilled.)

Total servers: 35

Serve as a FIFO queue, but "VIPs" enter the queue with a virtual **15** second wait (i.e. as if they had joined the queue 15 seconds earlier).

<u>Setup D</u> : (V-design)

Total servers: 35

??? - will be unveiled later.

<u>Question</u>: Consulting the following charts (2-5), which of the setups (A, B or C) would you recommend, if any ?

Chart 2 : 1000 Calls - ASA

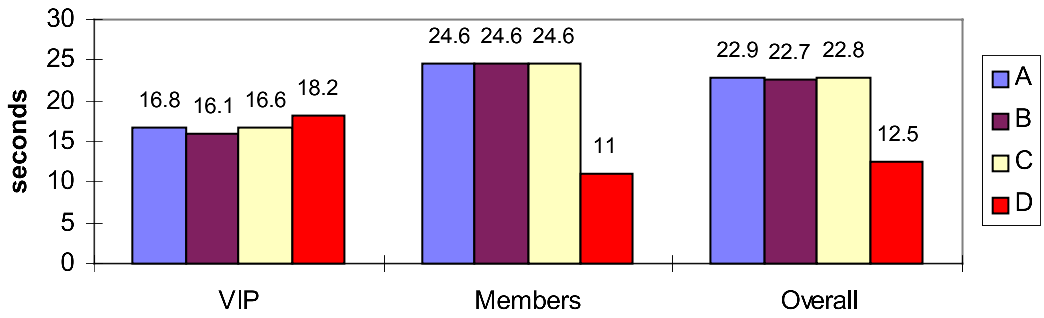


Chart 3 : 1000 Calls - Abandonment

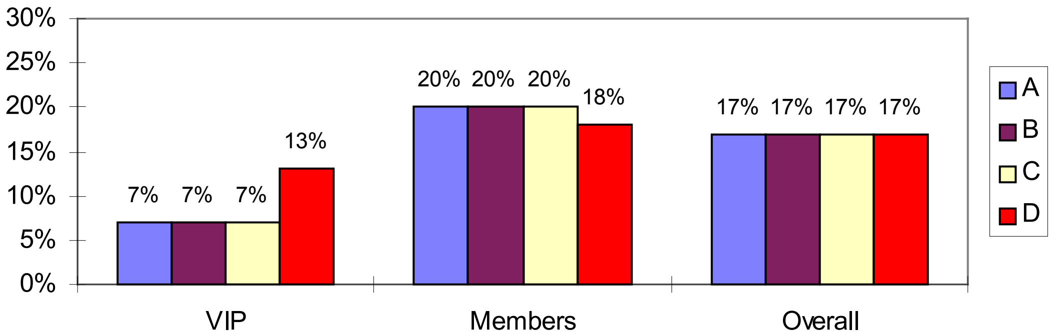


Chart 4 : 1000 Calls - Overflows

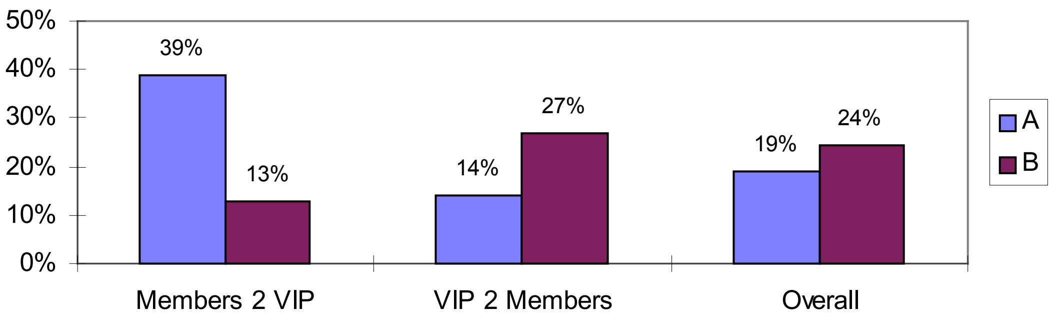
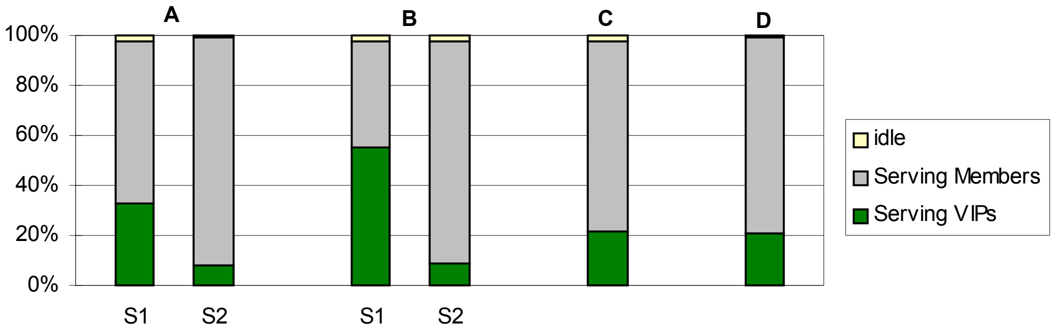


Chart 5 : 1000 Calls - Servers' utilization profiles



All three setups (A,B and C) yield practically the same service levels, with respect to both ASA and abandonment. At this point there is still not enough information to evaluate Setup C, but from the overflows (Chart 4) and utilization profiles (Chart 5) we see that Setup B is preferable to A – indeed B has less “VIPs” overflowing to “Member” servers, and the “VIP” servers spend more time on their pre-assigned “VIP” customers (it is reasonable to assume that overflows of the type “VIP 2 Members” are less desirable than “Members 2 VIP” – preferring to have the high priority “VIP” customers receive service from the pre-assigned “VIP” servers).
Note that the servers are all almost fully utilized (Chart 5) – this is in fact a heavy traffic scenario !

The managers of the call-center contemplated the alternative setups, realizing that additional experiments are required. This happened to take place shortly after the release of a new series of products, causing a 50% increase in avaerage call volume (i.e. 1500 calls per hour).

<u>Question</u>: Reconsider your recommendation in view of the additional experiments (Charts 6-9).

Chart 6 : 1500 Calls - ASA

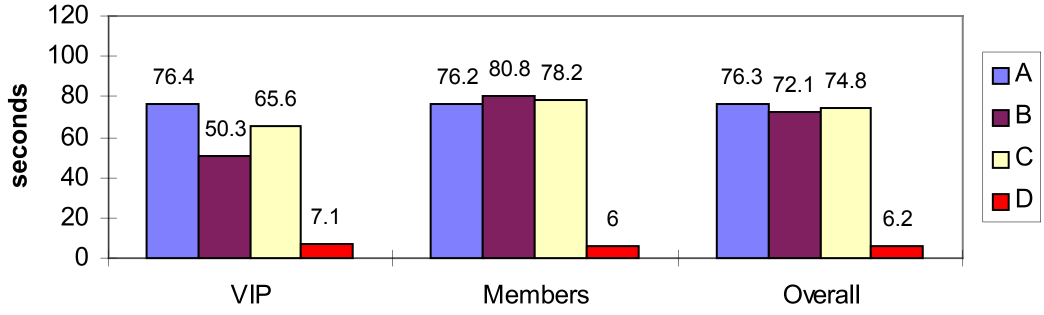


Chart 7 : 1500 Calls - Abandonment

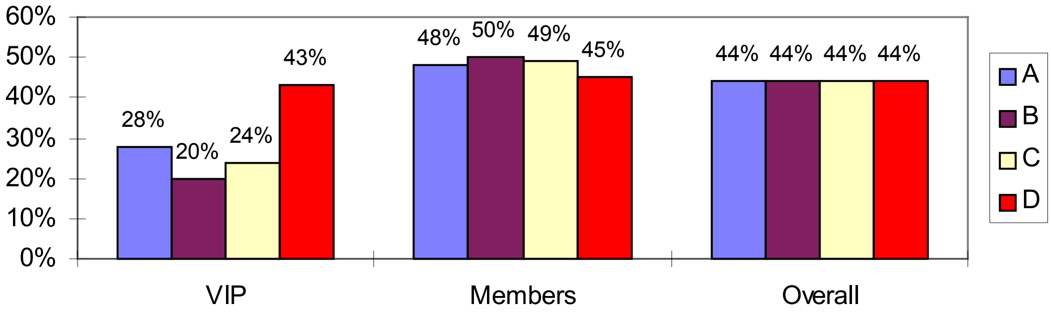


Chart 8 : 1500 Calls - Overflows

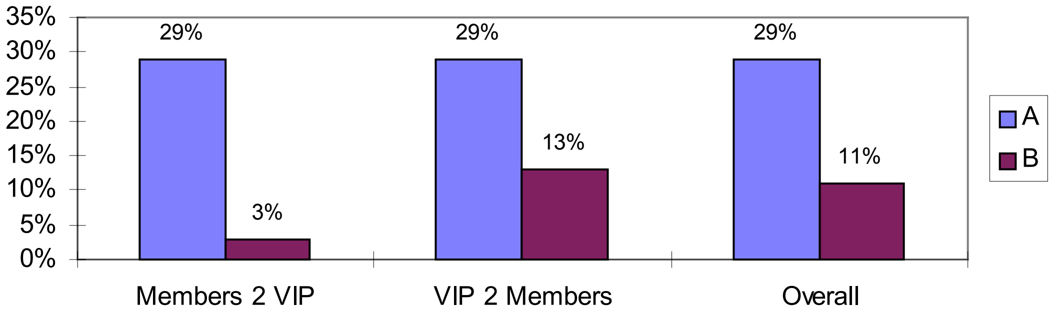
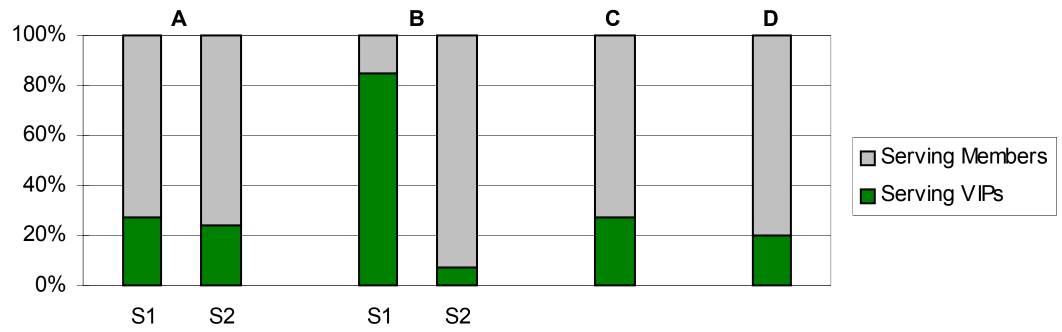


Chart 9 : 1500 Calls - Servers' utilization profiles



Now it is clear that Setup B is superior to to both A and C – for "VIPs", both ASA and abandonment are lowest, while the overall performance is at least as good as the other setups. As before, Setup B is better than A regarding the overflows and utilization profiles: under Setup A there is so much overflowing in both directions that both server types appear to be doing the same kind of work, as manifested through their very similar utilization profile; under Setup B, on the other hand, the utilization profiles demonstrate that each server type is mostly "dedicated" to his own customers, as should be the case with SBR. <u>One concludes that Setup B achieves skills-based routing with improved performance through limited mutual assistance !</u>

A non-work-conserving policy has the potential of reducing the amount of overflowing: the likelihood of having free servers to handle new arrivals of their <u>own</u> designated type is increased by limiting the cases in which calls of the <u>other</u> type are routed to them.

Note the superior performance of Setup D (ASA-wise). Also recall that, as a V-design, Setup D has no overflows. This apparently superior setup was NOT tested since no one had considered implementing such a *control policy* - a simple **LIFO** discipline !

<u>Question</u>: Can you explain the fact that under the LIFO discipline the ASA <u>decreased</u> as traffic <u>increased</u> ?

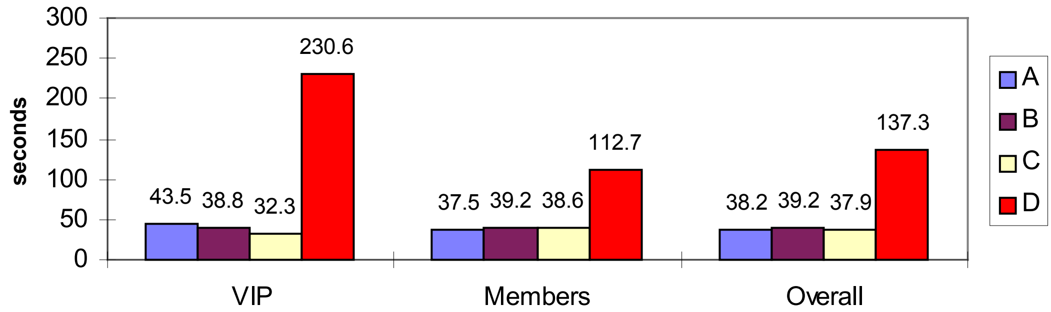
ASA = fraction queued of those served ×
average wait of those queued and served

With a FIFO discipline both the "fraction queued of those served" and the "average wait of those queued and served" increase with the traffic load. However, under the LIFO discipline the "average wait of those queued and served" <u>decreases</u> with the traffic load since customers arrive more rapidly and therefore the next one served <u>from the queue</u> (which was the last one in the queue to arrive) has spent less time waiting in queue. Customer "stuck" in the queue will eventually abandon but their long waiting times are not included in the ASA. As a result, when the traffic load is high and the "fraction queued of those served" nears 1 (and thus doesn't change much with the traffic load), the decrease in the "average wait of those queued and served" becomes dominant, thus the ASA decreases as well.

Judging by the service level, as measured at this call center (i.e. the ASA), Setup D yields the best results, therefore it seems that a LIFO discipline should be implemented.

<u>Question</u>: Why isn't it reasonable to use a LIFO discipline at our customer service call-center ? How can this conflict (with the apparent superiority of LIFO performance) be settled ?

Chart 10 : 1500 Calls - Average Speed to Abandon



From Chart 10 we see the "price" one pays for using Setup D : For those customers who eventually abandoned, not only did they not receive the service they wanted, but they waited a very long time in queue before giving up. Thus, the method of taking from the end of the line is "unfair" in that it literally abuses patient customers (i.e. those willing to wait a relatively long time.) Note that the abandonment profile of LIFO is also not favorable (Chart 7) – while overall it is similar to the other setups, there are many more "VIPs" abandoning under LIFO.

The "conflict" arises from the choice of ASA as the measure of service level. As demonstrated above, a call center can have very low ASA while providing disastrous service to a large part of their customers (in this case the 44% who abandon). In complex systems, the prevailing assumption that low ASA indicates good service overall is not necessarily true. It is therefore essential to include service measures regarding customers that did not receive service, for example the fraction of customers that abandon.

Remark: One can actually reduce the fraction of "VIPs" that abandon under the LIFO discipline. This can be achieved with variations on LIFO, in which the "VIPs" get higher priority by letting them enter the queue with a virtual negative wait (similarly to the variation that Setup C has on a FIFO discipline.)

A note on Optimality: We have not discussed *optimal* control policies for the X-design with multi-server pools and abandonment. The reason for this is simple – we just don't know what is optimal, or even approximately optimal! Not even for the much simpler N-design with single servers and abandonment; and not even for V-design without abandonment but with many servers. These issues, which are of great practical significance, provide ample challenges for current leading-edge research.

A note on Performance Measures: As part of the search for an optimal control, one must specify a criterion (performance measure) for optimality. This is significant – we already witnessed the consequences of a "wrong" criteria (e.g. ASA leading to the optimality of LIFO.) One would think that with abandonment there exists a natural definition for "operational optimality" – simply the fraction abandoning, perhaps weighted over types according to importance. But even here, some call-center managers would argue that those abandoning within say 3 seconds should not be included. More fundamentally, the overall optimality criteria actually becomes complex – for example, one should not ignore significant "costs" due to customers who abandon before being served (lost-opportunity costs, negative reputation costs,…).

Technical Appendix : Simulations - the computational effort

Many simulations were run to produce the graphs and charts appearing in this note. The code of these simulations was written in C, and uses the SSS library, written by Prof. M. Pollatscheck. Most of the simulations were run on a 133 MHz Pentium with 32MB RAM.

Following is a table with the number of batches in each simulation (listed by chart/graph), batch length (in minutes, according to "background story") and the simulation's approximate run time:

Chart / Graph	Batches	Batch Length	Run Time	Type
Graphs 1 and 3 and Chart 1	500	from 1500 (small ρ) to 20000 (large ρ)	from seconds (small ρ) to hours (large ρ)	B
Graph 2	1000	5000	minutes	A
Graphs 4-6	10000	10000	hours	A
Graphs 7-10	1000	from 1500 (high threshold) to 15000 (low threshold)	from minutes (high threshold) to hours (low threshold)	B
Graphs 11	1000	from 1500 (high threshold) to 15000 (low threshold)	from minutes (high threshold) to hours (low threshold)	C
Charts 2-8	1000	1500	hours	D

Types:

A - a single simulation produces all the data in the chart / graph.

B - each simulation produces one data point in the chart / graph.

C - multiple simulations run to produce a single data point in the chart / graph.

D - each simulation produces all the data for a single setup.