**NAME: DABO LAURA I.**

**MATRIC NUMBER: 16/SCI01/053.**

**COURSE CODE: CSC 410.**

**COURSE TITLE: COMPUTER SYSTEM PERFORMANCE EVALUATION.**

**ASSIGNMENT TITLE: ASSIGNMENTS FOR 400 LEVEL OF CSC 410.**

**START DATE: 15-04-2020.**

**END DATE: 30-04-2020.**

**QUESTIONS**

1. Explain the concepts of operational laws as applied to computer and network system performance evaluation.
2. Exhaustively describe at least eight operational laws that are widely employed in computer system performance evaluation.
3. Distinguish between the Forced Flow Law and the Residence Time Law from a systems perspective (not by definition).
4. Discuss some basic queuing models and basic queuing disciplines.
5. Discuss how to resolve some basic queuing problems.
6. You have been presented with some systems performance evaluation report; the first was done using measurement technique only, the second was done with simulation technique only, the third was done using analytical technique only, then the fourth was done using measurement and simulation technique only, while the fifth was done using simulation and analytical technique only and the sixth was done using measurement and analytical technique only. You are the director of information technology infrastructure in your firm and your firm is about to acquire the information infrastructure concerned in this report
7. What specific motive will you consider imperative in general?
8. Why do you consider this metric important?
9. If they are not in this report, what will you do in such a report?
10. If they are part of the report, what is the first action to take with that report and why?
11. Which reports or combination of report will you adopt and why?

**ANSWERS**

1. A number of laws are derived which establish relationships between throughput, response time, device utilization, space-time products and various other factors related to computer system performance. These laws are obtained by using the operational method of computer system analysis. The operational method, which differs significantly from the conventional stochastic modelling approach, is based on a set of concepts that correspond naturally and directly to observed properties of real computer systems.

Operational laws are simple equations which may be used as an abstract representation or model of the average behaviour of almost any system. One of the advantages of the laws is that they are very general and make almost no assumptions about the behaviour of the random variables characterising the system. Another advantage of the laws is their simplicity: this means that they can be applied quickly and easily by almost anyone.

Based on a few simple observations of the system the performance analyst can, by applying these simple laws, derive more information. Using this information as input to further laws the analyst gradually builds up a more complete picture of the behaviour of the system. Note that although we will talk in this section about operational laws in the context of systems, the laws are equally applicable to the observations obtained from models and we will have occasion to use the laws in this way later in the course.

The foundation of the operational laws are observable variables. These are values which we could derive from watching a system over a finite period of time. We assume that the system receives requests from its environment. Each request generates a job or customer within the system. When the job has been processed the system responds to the environment with the completion of the corresponding request.

1. Some of the operational laws that are widely employed in computer system performance evaluation are;
2. Utilization law

If we know the amount of processing that each job requires at a resource then we can calculate the utilisation of the resource. Let us assume that each time a job visits the *i*th resource the amount of processing, or service, time it requires is . Note that service time is not necessarily the same as the residence time of the job at that resource: in general a job might have to wait for some time before processing begins. The total amount of service that a system job generates at the *i*th resource is called the service demand,

The utilisation of a resource, the percentage of time that the ith resource is in use processing to a job, is denoted . The utilisation law states that

The utilisation of a resource is equal to the product of the throughput of that resource and the average service requirement at that resource.

Example: Consider again the disk that is serving 40 requests/second, each of which requires 0.0225 seconds of disk service. The utilisation law tells us that the utilisation of the disk must be 40 × 0.0225 = 90%.

1. Forced flow law

It is often natural to regard a system as being made up of a number of devices or resources. Each of these resources may be treated as a system in its own right as far as the operational laws are concerned, with the rest of the system forming the environment of that resource.

A request from the environment generates a job within the system; this job may then circulate between the resources until all necessary processing has been done; as it arrives at each resource it is treated as a request, generating a job internal to that resource.

Suppose that during an observation interval we count not only completions external to the system, but also the number of completions at each resource within the system. We define the visit count, , of the *i*th resource to be the ratio of the number of completions at that resource to the number of system completions

.

More intuitively, we might think of this as the average number of visits that a system-level job makes to that resource. For example, if, during an observation interval, we measure 10 system completions and 150 completions at a specific disk, then on the average each system-level request requires 15 disk operations.

The forced flow law captures the relationship between the different components within a system. It states that the throughputs or flows, in all parts of a system must be proportional to one another. In other words, it relates the throughput at the individual resources () to the throughput at the complete system (). It is stated as follows

The throughput at the *i*th resource is equal to the product of the throughput of the system and the visit count at that resource.

An informal interpretation of this law is that, since the visit count defines the number of visits to a resource or device that each job needs in order to complete its processing, the resource must keep up a correspondingly scaled completion rate to ensure that the system completion rate is maintained.

1. Little’s law

The best known and most commonly used operational law is Little’s law. It is named after the man who published the first formal proof of the law in 1961, although it had been widely used before that time. Little’s law is usually phrased in terms of the jobs in a system and relates the average number of jobs in the system *N* to the residence time *W*, the average time they spend in the system. Let *X* be the throughput. Then

The average number of jobs in a system is equal to the product of the throughput of the system and the average time spent in that system by a job.

Given a computer system, Little’s law can be applied at many different levels: to a single resource, to a subsystem or to the system as a whole. A little care may be necessary if the law is applied in this way, as the definitions of the number of jobs, throughput and residence time used at the different levels must be compatible with each other. At different levels of detail, different definitions of “request” are appropriate. For example, when considering a disk, it is natural to define a request to be a disk access, and to measure throughput and residence time on this basis. When considering an entire transaction processing system, on the other hand, it is natural to define a request to be a user-level transaction, and to measure throughput and residence time on this basis.

Example: Consider a disk that serves 40 requests/second (*X* = 40) and suppose that on average there are 4 requests present in the disk system (waiting to be served or in service) (*N* = 4). Then Little’s law tells us that the average time spent at the disk by a request must be 4/40 = 0.1 seconds. If we know that each request requires 0.0225 seconds of disk service we can then deduce that the average queuing time is 0.0775 seconds.

1. General response time law

One method of computing the mean residence or response time per job in a system is to apply Little’s law to the system as a whole. However, if the mean number of jobs in the system, *N*, or the system level throughput, *X*, are not known an alternative method can be used. Applying Little’s law to the *i*th resource we see that , where is the mean number of jobs at the resource and is the average response time of the resource.

From the forced flow law we know that. Thus we can deduce that

*Ni*/*X* = *ViWi*

The total number jobs in the system is clearly the sum of the number of jobs at each resource, i.e. if there are M resources in the system. We know from Little’s law that and from this we arrive at the general residence time, or general response time law:

The average residence time of a job in the system will be the sum of the product of its average residence time at each resource and the number of visits it makes to that resource.

Example: A web service running on an application server requires 126 bursts of CPU time and makes 75 I/O requests to disk A and 50 I/O requests to disk B. On average each CPU burst requires 30 milliseconds (waiting + processing time). Monitoring has shown that the throughput of disk A is 15 requests per second and the average number in the buffer is 4 whilst at disk B the throughput is 10 requests per second and the average number in the buffer is 3. Using Little’s Law we calculate the residence time at each of the disks (remembering that the number in the system is the number in the buffer +1):

Then

1. Interactive response time law

The name of this law dates back to the time when most of the systems which were being modelled were mainframes processing both interactive jobs and batch jobs. The think time, *Z*, was quite literally the length of time that a programmer spent thinking at his terminal before submitting another job. More generally interactive systems are those in which jobs spend time in the system not engaged in processing, or waiting for processing: this may be because of interaction with a human user, or may be for some other reason.

For example, if we are studying a cluster of PCs with a central file server to investigate the load on the file server, the think time might represent the average time that each PC spends processing locally without access to the file server. At the end of this nonprocessing period the job generates a fresh request.

The key feature of such a system is that the residence time can no longer be taken as a true reflection of the response time of the system. The think time represents the time between processing being completed and the job becoming available as a request again.

Thus the residence time of the job, as calculated by Little’s law as the time from arrival to completion, is greater than the system’s response time. The interactive response time law reflects this: it calculates the response time, R as follows:

The response time in an interactive system is the residence time minus the think time.

Note that if the think time is zero, and , then the interactive response time law simply becomes Little’s law.

Example: Suppose that the library catalogue system, has 64 interactive users connected via web browsers, that the average think time is 30 seconds, and that system throughput is 2 interactions/second. Then the interactive response time law tells us that the response time must be 64/2 − 30 = 2 seconds.

1. Bottleneck analysis

The resource within a system which has the greatest service demand is known as the bottleneck resource or bottleneck device, and its service demand is , denoted . The bottleneck resource is important because it limits the possible performance of the system. This will be the resource which has the highest utilisation in the system.

The residence time of a job within a system will always be at least as large as the total amount of processing that each job requires, this will be the time that the job takes even if it never has to wait for a resource. The total amount of processing that a job requires is D, the total service demand, . In general, there will be some contention in the system meaning that jobs have to wait for processing so the residence time will be larger than this, i.e.

The throughput of a system will always be limited by the throughput at the slowest resource (think of the forced flow law); this is the bottleneck device. By the utilisation law, at this resource, let’s call it *b*,. Therefore, since

It follows that if we wish to improve throughput we should first concentrate on this resource, improving throughput at other resources in the system might have little effect on the overall performance.

Using Little’s law or the interactive response time law, we can derive a tighter bound on the response time which applies when the system is heavily loaded (i.e. the mean number of jobs, *N*, is high). Applying the interactive response time law to the throughput bound, we obtain:

Applying Little’s law we obtain . Thus the asymptotic bound for residence time or response time is:

Similarly the bound on the throughput of an interactive system may be made tighter when the system is lightly loaded (i.e. the mean number of jobs, *N*, is small). From the interactive response time law:

Applying Little’s law (when Z = 0) we obtain X ≤ N/D.

Notice that the bottleneck depends on both resource parameters (Xi or Si) and the workload parameters (). If we change the number of visits that each job makes to a resource we might move the bottleneck.

1. Throughput Law: Throughput is equal to device utilization divided by the product of average device service time and the average number of requests per job for that device. Symbolically,

 for any *i*

An important aspect of the throughput law is that it is independent of a large number of factors which must normally be specified in other analyses. These factors include: the degree of multiprogramming; the service time distribution for each device and processor in the system; the fact that the system is or is not in statistical equilibrium; the fact that overlap of CPU and I/0 processing within a single program is or is not permitted.

In other words, the throughput law is valid regardless of the status of these other factors. For this reason, the throughput law can be regarded as a universal law of computer system performance. A probabilistic version of the throughput law, which is valid in the context of the central server model, was derived earlier by Buzen.

The throughput law bears certain similarities to elementary laws of motion. One point of similarity between this result and the throughput law is that distance, acceleration and time are most naturally regarded as operational variables in the above relationship. Note that it is not natural to consider them as expected values of random variables since no underlying probabilistic model is involved.

It is, of course, true that the mathematical argument required to derive the relationship between distance, acceleration, and time is significantly more complex than the argument required to derive the throughput law. However, the next few sections of this paper will demonstrate that operational laws of greater mathematical interest also exist.

1. Space-Time Product Laws

Space-time products are often used to evaluate program performance and assign accounting charges to programs in virtual memory systems. Essentially,

a program's space-time product is equal to its execution time multiplied by the average amount of memory allocated to it during its execution.

Since space-time products, response time, and throughput are all used as indicators of system performance, it is interesting to examine the manner in which these quantities are related.

Begin by considering an arbitrary system which may be operating in either an interactive or a batch processing mode. Let the operational variable A be defined as the number of jobs which arrive at the system during the observation interval.

which will be referred to as the space-time product throughput law, states that the throughput is equal to average amount of memory in use divided by average space-time product.

A similar relationship between average space-time product and average response time in the asymptotic case yields

and is referred to as the space-time product response time law.

1. The forced flow law captures the relationship between the different components within a system whereas in residence time law, there is one terminal per user and the rest of the system is shared by all users. This law holds even if the job flow is not balanced
2. A queuing system model can be defined as a representation that captures and quantifies the phenomenon of waiting in lines. The three basic elements within a queuing system are entities, servers (or resources) and queues (or waiting lines). Entities can represent either customers or objects. Servers can represent persons or production stations that treat or interact with the entity. A server is defined as an entity that can affect, or even stop, the flow of jobs through the system. In a computer system, a server may be the CPU, I/O channel, memory, or a communication port. Queues are the holding or waiting position of entities. A queue is just that a place where jobs queue for service. The queue size may be assumed to be either finite or infinite.

For example, the physical confines of a buffer area (i.e., queue) between two workstations along a production line may behave as a cap on the potential number of entities waiting for service; since historically we have knowledge that the maximal number of entities has filled on occasion this queuing area, we say the queue size is finite. In contrast, a call center may have the capacity to queue, or place on hold, up to 1000 incoming calls unable to be serviced by the three service representatives; since there is no history of the call center approaching its holding queue capacity, for all intents and purposes this queue can be assumed to be infinite.

1. The M/M/I queuing model: M/M/1 queue is the most commonly used type of queue. It is used to model single processor systems or to model individual devices in a computer system. The M/M/1 queuing model is characterized by a Poisson arrival process and exponential service time distributions, with one server and a FIFO queue ordering discipline. The system may represent an input buffer holding incoming data bytes, with an I/O processor as the server. A few of the quantities that we will be interested in for this type of queuing system are the average queue length, the wait time for a customer in the queue, the total time a customer spends in the system, and the server utilization.

Let's look at the exponential service distribution first. It is given as:

1. The M/M/I/K model: An interesting and realistic variation on the basic M/M/1 model is a system with a finite queue size. In this model, once the queue is full, new arrivals are lost and are never provided service. This is quite realistic, for example, in an input system with finite input buffer space and no flow-control protocol.
2. The M/M/C model: The M/M/C model consists of a single waiting line that feeds C identical servers. The arrival process is considered to be Poisson, and the servers have exponential service times.
3. The M/G/I model: The queuing models that we have discussed so far have all had the Markov property for arrival and service processes, making it possible to model the system as a birth-death process and to write the flow balance equations by inspection. Next, we will look at a system in which the service time does not have the Markov property. In the M/G/1 model, each customer has different and independent service times. Because service times are not guaranteed to be Markovian, the system is not representative of a Markov chain and we must resort to other methods to derive meaningful statistics. One approach commonly taken is to look at the process that describes jobs leaving the system, which is a stochastic process that also happens to be a Markov chain. It has been shown that, in the limit, the distribution for the number of jobs in the system at any point in time and the number of jobs in the system observed when a customer departs from the system, are identical. Summarizing the procedure, then, we can analyze certain aspects of the system that are described by a non-Markovian process by observing a Markovian subportion of the system (in this case the departure process) and extrapolating the results back to the original system. This type of analysis relies on what is known as an embedded Markov chain. The derivation of the statistics for the M/G/1 system is beyond the scope of this book. The general M/G/1 system is useful in many situations, because we can characterize a known service process in terms of its moments and then evaluate its performance in the presence of a random arrival process.
4. Queue discipline is the order that either customers or objects are selected from a queue to advance to receive service. Discipline of a queuing system means the rule that a server uses to choose the next customer from the queue (if any) when the server completes the service of the current customer. Commonly used queue disciplines are:
5. First-come, First-out (FIFO): A customer that finds the service center busy goes to the end of the queue. This discipline is also referred to as first-come, first-served (FCFS).
6. Last-In, First-Out (LIFO): A customer that finds the service center busy proceeds immediately to the head of the queue. It will be served next, given that no more customers arrive.
7. Preemptive LIFO (P-LIFO): After arrival to the system, a customer immediately enters service and pushes the customer in service, if any, back to the head of the queue. Customers may be pushed out of service several times before completing service and leaving.
8. Random Service: The customers in the queue are served in random order, meaning that when completed, the next customer to enter service is randomly chosen among the customers waiting in the queue.
9. Round Robin: Every customer gets a time slice. If its service is not completed, then it will reenter the queue.
10. Processor sharing: Whenever n customers (n W 0) are present in the system, the server processes all of them simultaneously, but at a uniform rate of 1/n each. This means that all current customers are in service.
11. Priority Disciplines: Every customer in the queue has a (static or dynamic) priority; the server selects always the customers with the highest priority. This scheme can use the preemption discipline.
12. Shortest Job First (SJF): When service is completed, the customer to enter service next is the customer in queue with the smallest remaining service time. If a customer arrives with a service time smaller than the customer in service, then the new customer enters service immediately, forcing the current customer to get back into the queue. As with P-LIFO, customers can enter service several times before finally completing service and departing.
13. Using queue theory can resolve some basic queuing problems. Queuing theory is a scientific approach to analyse waiting lines. The main objectives of queuing theory are to improve customer service and reduce the system operating costs.
14. Simulation technique.
15. I consider it important because analytical technique is a form of the simulation technique and measurement technique are not always possible if the system does not exist yet. It is flexible and dynamic tool that provides the modeller with the ability to define models of systems and put them into action. It provides a laboratory in which to study myriad issues associated with a system without disturbing the actual system. A wide range of experiments can be performed in a very controlled environment, time can be compressed and allowing the study of otherwise unobservable phenomena and sensitivity analysis can be done on all components.
16. Carryout a systematic approach.
17. The first action to take is to define the goal, load and metric. It helps to require knowledge of the system and its use.
18. The fifth report using simulation and analytical technique only. The reason is because analytical technique are often used to gain insight during a development phase and also to learn fundamental facts about a system while the simulation technique requires relatively lesser efforts and it is averagely accurate.