Evaluation Principles

8/15/07

These principles concern how computing systems perform under various computational loads and how much capacity they need to deliver their results on time.

The principal tools of evaluation are modeling, simulation, experiment, and statistical analysis of data.

A. Throughout computer science, there is a basic concern with the questions "Is it correct?" and "How long does it take?" The correctness question relies on the tools of logic and deduction to prove that the system or algorithm meets its specifications. However, the tools of logic are not as effective for the performance question; the best they do is relate computing time to the size of the input.

B. Few computations run in an environment that is completely insulated from other computations. Thus the answer to the responsiveness question depends on much more than the size of an input; it depends on the random interactions of many computations using the same servers and networks, and interacting in unpredictable ways. The fundamental quantities, such as numbers of arrivals, numbers of service completions, service times, response times, or queue lengths, are all random variables. We therefore turn to modeling, simulation, experiment, and statistical analysis to answer "How long does it take?" We express answers with mean values for throughput, response time, and queue length, possibly accompanied with confidence intervals centered on these means. We also in some cases look at percentiles, variances, and other quantities derived from the distributions of these random variables.

C. A model is a representation of a real system, usually with simplifying assumptions. Models are used not only to understand systems, but to calculate and predict performance quantities such as throughput and response time, and to calculate system capacity needed to meet performance requirements. The performance analyst's objective is to find accurate computational algorithms for prediction and capacity planning that are significantly faster than building and measuring the system. The analyst validates models by demonstrating through

simulation or experiments that their calculated performance quantities are close to the real values in the system.

D. The state of a model is a random variable, as are the events that trigger transitions among the states. This means that model inputs are presented as random processes, and outputs as mean values and other distributional quantities such as variances and confidence intervals.

E. Performance calculation means to use a model to calculate performance quantities from parameters that have been measured or assumed.

F. Performance prediction means to extrapolate parameter values to a future time period and then use performance calculation to estimate performance quantities for that period.

G. Three kinds of models used to calculate performance quantities are benchmark, simulation, and analytic. A benchmark is a suite of programs representing a typical workload; by running the benchmark and measuring its completion time, the analyst can compare different systems or configurations. A simulation is a model that imitates the real system by tracing system states that result from jobs requesting, obtaining, and releasing resources; performance quantities are measured by counting events and adding up the holding times of states entered by those events. An analytic model sets up equations whose solution gives the performance quantities for the system.

H. In the early 1900s, telephone engineer A K Erlang applied mathematics to compute how much capacity (including switchboard operators) a telephone exchange must have to assure a reasonable level of service. Through measurement, he discovered that the times between moments when someone picks up a phone (arrivals) are exponentially distributed, and that the lengths of phone calls (marked by subsequent completions) are also exponentially distributed. He applied this to a simple mathematical model with “states” representing the number of phone calls in progress and “transitions” corresponding to arrivals and completions. He was able to solve for p(n), the equilibrium probability that n calls are in progress. He could then size a telephone exchange by choosing N (the upper limit on calls in progress) so that p(N) is below a desired threshold (for example, 1% chance of loss, or 99% availability). Erlang’s basic methods underlie many modern analytic models for system performance.

Computing systems can be represented as sets of equations balancing transition flows among states.

A. Computing systems are discrete state systems. As individual computations move through their internal states, they dynamically change their demands for resources. Internal state changes of computations trigger changes in the computer system state.

B. In the telephone exchange example, the individual act of initiating a call triggers a system state change from n to n+1 calls in progress; and of completing a call triggers a system state change from n to n-1 calls in progress. Thus the arrival rate pushes the system toward higher states and the completion rate toward lower states.

C. A computing system could in principle be completely and accurately described by its states and by a matrix that tells the transition rate between every pair of states. The states and transition matrix define a system of equations with unknowns p(n), the probability of the system's being in state n. These equations can be used to find transient or steady state solutions. A transient solution means that p(n) is actually a function of time, and thus the equations help determine p(n,t+1) from p(n,t). A steady-state solution means that, after enough time, there is little change in the solution, and thus the dependence on t can be ignored. Any performance quantity can then be calculated by summing, over all states, the quantity for each state weighted by the state's probability.

D. Steady state solutions work well when the system is not overloaded and its parameters are time invariant, meaning that average lengths of queues do not grow ever larger as time increases. Because steady state solutions are easier to compute and usually validate well, they are commonly used in performance analyses.

E. Unfortunately, a full state space representation is impossibly large for most real systems, which have enormous numbers of states. Although a system of equations over all the states is a perfect representation, its computational infeasibility makes it unattractive as an alternative to building and measuring the real system. However, with a few simplifying assumptions, these equations can be significantly simplified and solved.

F. To make the p(n) solutions computationally feasible, the modeler makes various simplifying assumptions. For example, the modeler may assume that the number of transitions into any state equals the number out (a balance assumption equivalent to assuming steady state). Or, the modeler may assume that internal events in computations are never simultaneous, implying that transitions of

more than +1 or -1 to a state component will not be observed. The simplifying assumptions lead to a much simpler transition matrix whose equations can be solved rapidly.

Network of servers is a common, efficient representation of computing systems.

A. Most computer systems are configured as networks of servers with interconnection pathways. A "job" is a unit of computation created in the system by a user. A job moves from one server to another as it receives service. A job may not enter service immediately on arriving to a server because other jobs are queued up before it. In more complex cases, such as a multithreaded computation, one computation creates many jobs that move through the system in coordination with one another.

B. The state of such a system is a list n=(n1,n2,...,nK) of the number of jobs queued at each server. The system is closed if the total number of jobs is fixed (they just move around in the system). The system is open if new jobs can enter or old jobs can exit.

C. State transitions are triggered by a job moving from one server to another. These simple moves cause simple local changes in the state. For example, a job moving from server 1 to server 2 enters the new state (n1-1,n2+1,n3,...,nK). In most systems only one job moves at a time: state transitions are only of this form.

D. Two simplifying assumptions are usually made to reduce this system to a computationally feasible form. One is that the number of transitions entering every state equals the number exiting. This is usually a good approximation. The other is that transition rates depend only the individual server's queue and not on any other queue. Thus the rate of jobs moving from server 1 to server 2 may depend on n1 but not on n2 or any other n.

E. These assumptions lead to an algorithmically simple expression for p(n). The algorithm rapidly computes utilization, throughput, mean response time, and mean queue length. Practical experience shows that these models are accurate enough to be highly useful.

F. We have described a "single workload" model -- all jobs have the same parameters. However, many real systems have multiple job loads with different parameters. For example, scientific computations have high CPU demand and low disk demand, while database transactions have low CPU demand and high disk demand. Multiple workload models permit two or more job classes, each with its own set of parameters. They yield to similar analyses, although their computational algorithms

are slower. Multiple workload models enable performance prediction separately for each job class, for example, the response time of scientific jobs versus transactions.

Network-of-server systems obey fundamental laws on their utilizations, throughputs, queueing, response times, and bottlenecks.

A. A fundamental law is a relationship among performance quantities that holds in every system and every observation period.

B. The utilization law says that a server's utilization is the product of its mean service time and its throughput.

C. Little's law says that the mean queue length at a server is the product of its mean response time and its throughput.

D. The forced flow law says that ratio of server to system throughputs is a constant representing the number of visits a job requires of that server.

E. The memory space-time law says that the total amount of memory is the product of the system throughput and the mean memory space-time per job.

F. The total demand for a server is the product of visit ratio and mean service time per visit. The server with highest total demand is the bottleneck. That server is the first to saturate at 100% utilization as the total load on the network increases. Once that server saturates, the other server throughputs are determined by the visit ratios and cannot even attain 100%.

G. These laws all work for open or closed systems. If the system is closed, additional laws hold, such as the response time depending only on the total number of jobs in the network, the average user think time per job, and the system throughput.

Resource sharing, when feasible, is always more efficient than partitioning.

A. In many systems, individual jobs make requests for varying amounts of each resource (e.g., pages of memory or seconds of CPU time). The system allocates the requested amounts by withdrawing them from pools. If a request exceeds the amount still in the pool, the system will place the job in a waiting queue until the requested amount is available.

B. In this case, the total of a resource needed is the sum of the random variables representing each job's demand. The probability P(R) that the sum is less than a threshold R is the probability that a job will not wait when making its next request, given that the system has R units of that resource initially in the pool.

C. Partitioning is an alternative to pooling. The system allocates each job a fixed amount and the total of the fixed amounts is R. In this case, the probably that no job has to wait is significantly smaller than when they draw from a common pool. (Thus if Pi is the probability job i does not wait in its partition, then P1*P2*...*PN < P(R).) Moreover the waste (difference between resource actually needed and the allocated) is significantly smaller in the shared case than in the partitioned case.