16 Parameterisation and Workload Characterisation

So far in this course we have been concentrating on constructing a good representation of the system, appropriate for the investigation we wish to carry out. However it is not just the physical system which is represented within the model but also the effect of the environment upon the system. The most obvious influence of the environment upon the system is to produce *work* for the system to do. Therefore it is just as important to have a good representation of the workload of the system as to have a good representation of the system itself. Indeed performance measures are not, in general, absolute for a given system: performance is predicted on the basis of a given workload. This was reflected in lecture note 14 when validation of the input parameters was given equal importance with validation of the assumptions used within the behaviour of the model. Finding suitable and/or realistic values for input parameters is often termed *workload characterisation*.

There have been well-documented problems arising from badly parameterised models. The Hubble Space Telescope was rigorously simulated prior to its launch, as a cheaper alternative to testing. However, once positioned it was found that it could not produce a sharp image because the main mirror was badly flawed. The error was traced to the erroneous input data used in the simulations. Some corrective optics have now been added to the telescope in space (at a cost much higher than ground-based testing of the mirror), but the telescope will never perform as well as planned.

16.1 Parameterisation

Parameterisation is the process of assigning values to variables within a model in order to ensure that it is as accurate as feasible or appropriate. In a simulation model as well as the actual values to be used, there will be some consideration of the appropriate distribution to be used for generating the values. It is important that we consider the availability of data to assist in parameterising our models from the outset of model design and construction. For example, the level of abstraction at which work is represented must correspond to the level of abstraction at which the system is represented; i.e. since values must eventually be assigned to all input parameters, it would be a waste of effort to develop a very detailed model if there is not data available from which to construct a similarly detailed workload characterisation. Conversely, there is no need to invest a lot of effort in sophisticated statistical analysis of workload data to parameterise a crude model which is only intended to give rough estimates.

The first step is to choose which parameters to include in the model. As usual this will be influenced by the objectives of the study, and by consideration of what is likely to affect performance. If we examine the workload of a system it is likely to have many different characteristics, such as inter-arrival time of jobs, type of job, resource required per job, or size of job. However not all of these will have an impact on the performance of a system. For example, if we are modelling a router, within a communication network, at which the packet size does not affect the packet forwarding time then this characteristic of the workload (packets) does not need to be represented.

In the models we have considered during the course we have usually chosen the parameters implicitly at the same time as choosing the appropriate level of abstraction, and we have generally assumed a *homogeneous workload*, i.e. that all jobs or customers within the

system behave identically. Assigning values to parameters has also been simple within our models as they have not been based on real systems. However in practice a considerable part of the effort of a modelling study might be devoted to choosing appropriate values for input parameters.

Of course the model will also contain parameters which represent aspects of the system behaviour. Many of these will be *static* parameters which contain information about the system configuration. For existing systems these are often available from published materials (manuals, specification documents, webpages etc.). In contrast *dynamic* parameters usually characterise information about the working system which must be extracted or predicted from records produced during system operation. Moreover, the actual parameter used in a model may not directly match what is measured, but may need to be derived from static and dynamic measures.

For example, in a PEPA model of an application program, access to memory is represented as a single activity, with time delay t_{avg}. Direct measurement of this time on a real machine, however, could be problematic because the detailed steps of the memory hierarchy are not typically observable. However it is easy to see that the expression

$$t_{avg} = ht_c + (1 - h)t_m$$

represents the variable, where t_c is the time to access cache and t_m is the time to access main memory, whereas h is the hit ratio. The values of t_c and t_m will be available from the manufacturer's specification, and only h must be derived from measurements.

16.2 Measurement and Monitoring

Most approaches to measurement gathering are based on some notion of *event*. An event is a pre-defined change in system state—dependent on the metric being measured—which triggers the recording of data. Example events might be memory references, disk accesses or network communication operations.

The measures or *metrics* that a performance analyst might want to record can be classified as follows:

Event-count metrics For this class of metrics, what is being measured is quite simply the number of occurrences of some type of event. Examples might be the number of cache misses, or the number of disk I/O requests made by a program.

Secondary-event metrics Here the occurrence of an event stimulates the recording of some other system parameters which are used to define the metric of interest. For example, if we are interested in the number of messages in a buffer, the occurrence of an `enqueue` event or a `dequeue` event will stimulate the recording of the current queue length.

Profile These are not so directly related to individual events, but aim to build a complete picture of the overall behaviour of a program or system by an aggregate measure. In this case, a wide variety of events may be used to initiate the recording of data.

Dynamic information about the operation of a system under a workload can sometimes be extracted from the information recorded for general purposes such as accounting and

logging. However, in general the information required for performance analysis is more sophisticated and requires specific monitoring and measurement of the system.

Software performance monitors analyse the behaviour of the system during operation and write records describing the resource usage and the performance status of the system. For example, at specified intervals, queue lengths or device state indicators may be sampled and the results written to the record. Alternatively, certain events which are considered to be significant (such as swapping a page) may be documented in the record. In general the volume and form of the data recorded by software monitoring means that it is not possible to use it directly to parameterise the model. Instead it must be processed and analysed to produce suitable summaries. Most software monitors will include a reporting component which will undertake at least the initial stages of this task.

Hardware monitors are also available for some systems, and have the advantage of being "external" to the system under observation: they do not perturb system operation. However, in general the type of information which they can record is less detailed and so favour a homogeneous view of workload.

Some commonly occurring subsystems, such as databases, have given rise to specialised application software monitors. This is necessary because the subsystem has a certain amount of autonomy from the host operating system which makes access to its internal behaviour difficult.

There are four basic strategies for collecting measurement data. Each has advantages and disadvantages, and in particular the analyst should be aware of the extent to which the chosen strategy perturbs the system under observation.

Event-driven This strategy is best-suited to event-count metrics and secondary-event metrics for which the event frequency is low. Every occurrence of the event causes data to be recorded. For example if the desired metric is the number of page faults that occur in the execution of a program, the performance analyst can modify the page-fault-handling routine in the operating system to increment a counter whenever the routine is entered. An additional mechanism must also be provided to dump the contents of the counter when the program finishes execution.

This strategy has the advantage that the monitoring overhead is only incurred when the event of interest occurs. However if the frequency of the event of interest is high the overall perturbation of the behaviour of the system will be substantial.

Tracing This is essentially an enhanced version of the above strategy in which in addition to recording simple measures at each event, details of system state are also recorded so that each event can be identified. Thus in the example above, for each page fault we would record the address which caused the fault. This increases the time overhead of the strategy and can, additionally, lead to storage problems when the event of interest is frequent.

Sampling This is a general statistical measurement technique whereby a subset of the members of a population being examined is selected at random. It assumes that since the subset is chosen at random, its characteristics will approximate those of the total population.

In practical terms this means that at fixed time intervals a portion of the system state, necessary for calculating the metric of interest, is recorded. In this case the

overhead incurred is independent of the event frequency and is instead a function of the sampling frequency. The sampling frequency will depend on the resolution needed to capture the events of interest, i.e. if the event happens only rarely far more samples are likely to be needed. Furthermore, each run of a sampling-based experiment is likely to produce a different result since the samples occur asynchronously with respect to the system's execution. Nevertheless, while the exact behaviour may differ, the statistical behaviour should remain approximately the same.

Indirect An indirect strategy must be used for measures which are not directly measurable. The general approach is to define an alternative metric from which the metric of choice can be deduced or derived. In these cases the appropriate metric and measurement strategy will often need to be defined on an ad hoc basis and will depend on the ingenuity and creativity of the analyst.

16.3 Workload characterisation

Workload characterisation is the process of selecting the workload or workloads on which to base the performance study. Difficult questions arise even in considering an existing computing environment: What constitutes a "typical" workload? How should a measurement interval, or *time window*, be selected? Should data from several measurement intervals be averaged? These uncertainties are compounded in considering an environment that cannot be measured directly, for example in contemplating the movement of an existing workload to a new system, or the introduction of a new workload to an existing system. This problem is exacerbated by users who will often change their usage patterns dependent on the service available.

The workload has to be broken down into *workload components*. These are the significantly different tasks the system may be required to undertake. The choice of workload components can have significant impact on the results of a performance study. For example, if the packets in two networks are generally a mixture of two sizes—short and long—the workload to compare the networks should consist of short and long packet sizes. Using the wrong workload components will lead to inaccurate conclusions.

In most of the models we have considered in the course there has been only one workload component but most systems have many more. For example, a workstation in an academic environment might need to address all the following workload components: compilers, editors, file utilities, communication, scientific computation, graphics, basic services, text processing, and more. However for most performance studies, based on the objective of the studies we can abstract details of the workload components down to a minimal set. For example, if we wished to study the impact of scientific computation on communication we might consider a model with just three workload components: `scientific computation`, `communication` and `other`. If we were developing a queueing network model each of these workload components would be represented a distinct class of customers. For each class we would then need to know the service demand of customers of that class at each of the resources in the system, as well as the number of such customers (for a closed class) or the arrival rate (for an open class). If we were developing a GSPN model each class of customer would be represented in distinct cycles within the model (only closed classes can be represented in a GSPN) competing for the resources via synchronisations (cf. the

reader-writer model).

Often the resources we are interested in within our systems are quite low level, e.g. CPU and disks, whereas our workload components are expressed in terms of user applications. We must therefore derive the *devolved workload*. This involves viewing the system as a series of layers each built one on top of another. A layer offers services to the layer above and generates workload to the layer below. The devolved workload is then the amount of workload requested from the lower layer in order to satisfy one unit of workload from the layer above. For example, this technique might be used to derive the typical usage of a communication gateway involved in one `ftp` connection.

Most systems display varying behaviour over time although the time periods may differ. For example, a web server which presents pages offering advice on filling tax returns will exhibit varying patterns of use over a year, whilst most human-centric systems, such as a bank ATM, will have a workload which varies throughout each 24 hour period. In most cases, the time window which exhibits peak use is chosen for performance studies on the understanding that performance will only improve during the rest of the cycle.

When the objective of the performance study is to predict the behaviour of a system under a future workload, direct measurement of current workload characteristics is merely the starting point. The parameter values in the model will be based on a workload model which is itself predicted from the measured data. There are two major ways in which this prediction can be carried out.

Trend Analysis If historical workload measurements exist, the current workload measurements may be considered in relation to these and examined for trends. *Moving averages* and *exponential smoothing* are both techniques for extrapolating future values from a series. In this way the expected future values of the workload parameters are predicted.

Key Value Indicators Users or system managers often find it easier to predict the future in terms of their business than in terms of the service demands or load intensity of the resulting computer system workload. The key value indicator (KVI) is a quantifiable business variable such as the number of customers. The current workload is then related to the current value of the KVI and the users' predicted future computing needs in terms of KVI are then related back to give future workload estimates.

As explained in the previous section monitoring software or hardware can produce substantial amounts of information about the workload on the system. Parameterisation should have identified the key characteristics of that data for this model and modelling study. There are then various techniques which can be used to extract the necessary values from the data. For Markovian models, where we have already made an assumption that all delays within the model will be exponentially distributed, the analysis will generally be less sophisticated than what would be used to find input values for a simulation model. Commonly used techniques include the following:

Averaging Based on the sample data the *mean* value is calculated. In addition to the mean, the *median* and the *mode* are also sometimes useful ways of characterising a particular parameter as a single value.

Finding variability Variability is sometimes represented by the *coefficient of variation* which is the ratio of the standard deviation within a sample to the mean. For simulation models, this can then be used to fit a distribution to the sample points. The corresponding parameter in the model will then be drawn from that distribution with appropriate parameters.

Single parameter histograms A histogram shows the relative frequencies of various values, or range of values, of a parameter. Some simulation packages allow this sample data to be stored and used to provide varying values for the corresponding parameter during model execution via a look-up table.

Multi-parameter histograms Single parameter histograms fail to capture any correlation which may exist between different parameters of the workload. For example, short jobs may create a lower number of disk I/O requests and take a smaller amount of CPU time than long jobs. However, considering the parameters *number of disk I/O requests* and *CPU time* separately might allow jobs in the model with short CPU time and a high number of disk I/O requests—something not generally possible in reality. Considering multiple parameters at once attempts to overcome this problem. In practice correlated input parameters should usually be represented as functions of a single parameter.

Cluster analysis Most large computer systems have workloads consisting of several identifiable components or classes as in the packet size example above. There are two key goals when the observations of the system are analysed to identify these classes. Clearly, customers or jobs within a class should exhibit similar patterns of behaviour within the system. But also, if the objectives of the performance study require separate performance predictions for different workload components, they must be represented in different classes.

Cluster analysis is the principal way in which components or classes are identified within workload data. This technique is described in detail below.

16.3.1 Cluster Analysis

The data about a system generated by software monitoring may contain several thousand "user" profiles. The objective of cluster analysis is to reduce this data to a few key characteristics, identifying the different types of users in the system. Users who share characteristics are gathered together into clusters; eventually each cluster will be represented by one class in the model. The steps of cluster analysis are as follows:

1. Sample the workload data.
2. Select important workload parameters.
3. Transform parameters, if necessary.
4. Remove outliers.
5. Select a distance measure.
6. Perform clustering.
7. Interpret results.

8. Change parameters, or number of clusters, and repeat steps 3 to 6.
9. Select representative components from each cluster.

If there are n parameters of interest for each sample point (user observation) we can imagine that the selected sample points are mapped into n-dimensional space; for each sample point its position along the i axis represents its value for the ith parameter. The objective of cluster analysis is to form groups of sample points which are related with respect to their parameter values. In other words points which are close together spatially. The variance between points can be measured as their distance apart so the objective becomes to identify groups such that the variance within groups is minimised while the variance between groups is maximised.

Hierarchical approaches to cluster analysis are often taken and these can be *agglomerative* or *divisive*, depending on whether they start with as many different clusters as sample points and successively merge them, or whether they start with a single cluster containing all sample points which is successively split.