**Analytical Modeling/Analytical performance models**,

While not popular for microprocessors, THEY are suitable for evaluation of large computer systems. In large systems where details cannot be modeled accurately, analytical modeling is an appropriate way to obtain approximate performance metrics. Computer systems can generally be considered as a set of hardware and software resources and a set of tasks or jobs competing for using the resources. Multicomputer systems and multiprogrammed systems are examples.

Analytical models rely on probabilistic methods, queuing theory, Markov models, or Petri nets to create a model of the computer system.

A large body of literature on analytical models of computer exists from the 1970s and early 1980s. Analytical models are cost-effective because they are based on efficient solutions to mathematical equations. However, in order to be able to have tractable solutions, often, simplifying assumptions are made regarding the structure of the model. As a result, analytical models do not capture all the detail typically built into simulation models. It is generally thought that carefully constructed analytical models can provide estimates of average job throughputs and device utilizations to within 10% accuracy and average response times within 30% accuracy. This level of accuracy while insufficient for microarchitectural enhancement studies, is sufficient for capacity planning in multicomputer systems, I/O subsystem performance evaluation in large server farms, and in early design evaluations of multiprocessor systems. There has not been much work on analytical modeling of microprocessors. The level of accuracy needed in trade off analysis for microprocessor structures is more than what typical analytical models can provide. Experts used a Markov model to model a pipelined processor, or probabilistic techniques to processor a multiprocessor composed of superscalar processors. Queuing theory is also applicable to superscalar processor modeling, as modern superscalar processors contain instruction queues in which instructions wait to be issued to one among a group of functional units.

**Workloads and Benchmarks** used for performance evaluation of computers should be representative of applications that are run on actual systems. Contemporary computer applications include a variety of applications, and different 11 benchmarks are appropriate for systems targeted for different purposes.

**Tools and techniques of measurement**

Two approaches, known as the **stimulus approach** and the **analytical approach** to performance measurement are found in practice. In the **stimulus approach**, the system is treated as a black box, with a limited number of known functions. Simulated inputs (stimulus) are supplied to the black box and the response of the black box (outputs) is measured. A benchmark is a record of the systems response to a controlled work load.

In the **analytical approach**, the system is separated into parts for detailed measurement of internal behaviour. At times the two approaches are combined. Both involve measurement, and both can be used at any level in the object hierarchy. Three data transfer paths exist between a target process and a monitoring process

1. Information obtained using software tools is either passed to an internal monitoring process or to an external monitoring process via hardware probes. The latter is a hybrid tool.

2. Hardware event information is collected with hardware probes. This information can be recorded, or transformed using counters and comparators (a hardware tool).

3. Information can be fed back, from a monitoring process to a target process, for use in the adaptive control of the operating system. Control information is used to control measurement experiments. The data paths used by a monitoring tool to collect information, and whether the monitoring process is internal to the target computer or in a separate monitoring computer, classifies the tool into one of the three major categories: Software, Hardware or Hybrid. Firmware tools are often considered to be a separate classification, but they generally form part of a tool in one of the other categories.

**A measurement tool can be divided into a few conceptual sections**.

The **sensor section** is the interface between the target process and the measurement tool. It is the front end of the measurement tool. The sensor detects events of interest and measures the magnitude of the quantities being monitored. Sensors are often referred to as probes. A software probe is usually a sub-program, inserted into the target process, and a hardware probe is usually a set of connectors, terminated to the back plane or special test points. Sensors operate in one of two modes: internally driven or externally driven, determined by how the action of sensing is initiated. Externally driven tools usually sample system state in response to the occurrence of an event outside the target system, for example at the end of time periods measured by a clock. Data collected in this way has to be analysed statistically to produce meaningful results. Internally driven tools usually detect events occurring inside the target system, for example a procedure call or an interrupt vector. Thus, the data they collect is synchronized to the internal operation of the target process.

Data reduction occurs in the **transformer section**: Typically, the sensor produces a continuous stream of event descriptors and stimulus information. Only a subset of this observable information is of interest in any one measurement experiment. The transformer selects the subset of measured data to produce the set of measurements relevant to the experiment. Comparators are used to select events of interest from the event stream. These selected events are either recorded or used to trigger the recording of other information, often by mapping the event descriptor to a structure of data pointers. Counters are used to count events. Counter outputs are recorded, and can also be used to trigger the recording of other information. Time stamps can be added to event records and stimulus records. Data collected by the probes is either in the form of single bits (flags) or in functional groups of parallel signals (words). The operations performed by the transformer section on this data fall into the following categories:

* Data can be stored without change.
* Data can be masked to remove unwanted bits before storage.
* Data can be compared to reference values, and then flags; representing: equal to, less than, greater than, within range, outside range; are stored.
* Data can be logically manipulated by a function generator, for example two signals ANDed, before storage.
* Sequences of data patterns can be detected and stored.
* Sequences of data patterns can be detected and used to initiate storage of selected data sets which occur either before, during or after the sequence.
* Successive data inputs can be compared and the results stored.
* All of the above can be counted, i.e. counting the occurrence of specific data patterns.
* The time period between the occurrence of specific data patterns, in all of the above, can be measured.
* Some of the above can be combined to produce more complex reduction schemas. The resultant set of information (event trace and stimulus information) is stored in a data base. Small quantities of data are held in the tools memory and larger quantities are held on back-up store. The creation of this data base is an important function of the transformer.

The **analyser section** processes the data stored in the data base to produce the final output of the experiment, for example tables and graphs.

These outputs, plus the data base information are displayed by the **indicator section**. The analysis to be performed upon the data is determined by the hypothesis the experiment was designed to test.

Analysis often takes place at a later time using recordings, on tape, of several experiments. However, there are significant advantages in being able to analyse the data, in real-time, as the experiment progresses

• Analysis may indicate a flaw in the measurement specification, or inadequacies in the filtering done by the transformer, enabling these to be modified, thus, curtailing useless measurements.

• Analysis may indicate an error in the hypothesis, confirm the hypothesis, or indicate the need for further experiments. The experiment can be modified and re-run on the basis of the analysis. Thus, real-time analysis will speed up the evaluation process.

• Data can be used to interactively test and debug programs.

• Analysis results can be used to dynamically tune the operating system.

**Measurement Tool Characteristics**

Any measurement instrument can be described in terms of a set of characteristics typical of that class of tool. A set of characteristics has been developed for performance monitors. Introducing a tool to a system can impact upon the operation of the system. Interference can occur in a number of ways:

• Incorrect connection of hardware probes can degrade signals causing apparent hardware faults.

• Software tools use memory space, reducing the space available to the target process.

• The execution of a software tool uses system resources, causing a degradation in the actual performance and introducing inaccuracies into the measurements.

Improperly designed software probes can change the operation of the target process, introducing logical errors into the target process.

Two measurements are used to discuss the accuracy of a tool. **Precision** which refers to the number of digits available to represent data. **Resolution** refers to the maximum frequency at which events can be detected and correctly recorded.

The **scope** of a tool categorizes the classes of events which the tool can detect. For example, a time counter used to count CPU time, when the CPU busy signal is set, has a very limited scope. On the other hand, a logic-state analyser can be used in a variety of situations, and thus, it has a wide scope.

The **pre\*reduction capabilities** of the tool determine the type and size of experiments for which it can be used. Measurement can be limited by the number of comparators, the number of counters, the size of the plugboard, the available logic functions in the transformer, the number of signal connections, the size of the data storage memory, and the availability of signals for event detection. Lack of suitable event detection, signal recording, and pre-reduction facilities results in loss of information.

**Ease of Use** has a big impact upon the acceptance of a tool by analysts. Important considerations include: quality of the documentation, interactive setup of the transformer, ability to define events of interest, degree of difficulty of probe insertion (ease of installation), ability to activate and deactivate the recording of events, methods of accessing the data base, and the power of the analysis tools.

**Compatibility** refers to the matching of signal levels etc. at the interface between the target system and the 'monitoring system. Electrical voltage levels must be the same and hardware probes must not load signals. Software probes must not violate system protection mechanisms or interfere with the operation of the operating system.

As with other measurement tools, you get the features you are willing to pay for. The **cost** of purchase, installation, usage, expansion and maintenance of a tool should be compared to the expected cost savings due to performance improvement.

**Hardware Tools and Techniques**

A hardware tool consists of additional hardware added to the target system to collect signals of interest, and external transforming and analysis logic. Probes are connected to hardware signals, normally on the back plane, so that the activity of these signals can be monitored. These signals are fed to the transformation logic, where the subset required for analysis and display are filtered out. Hardware tools are classified according to their flexibility, due to the method of implementation, and the power of their transformation logic. **Fixed** hardware tools are completely hard wired. They are designed to measure specific parameters and are often incorporated in the initial design of the machine. In the latter case they are called **internal tools**, in comparison to external tools which are added and removed as needed. **Timing meters** and counting meters are typical fixed hardware tools. **A timing meter** measures the duration of an activity, for example - channel busy, by sampling the state of a signal associated with that activity. Thus, timing meters can be used to measure execution time and utilization. **Counting meters** count the occurrences of events, for example - count all the references to a memory bank. Counting meters can be used to measure execution frequencies and throughput. Sometimes, the event to be counted is generated by an **event trap**, a device which detects the occurrence of an event and generates a pulse to the counter, for example - detection of subroutine calls. The information accumulated by a meter can usually be read by an operator from a display, and in some cases, it can be read by a program. The simplest fixed hardware tool is the CPU wait light.

**Wired-Program hardware tools** include a logic plug-board that allows a variety of Boolean and counting functions to be implemented in the transformer section. Event filtering can be changed by rewiring the plug-board. In some advanced systems associative memory is also used to detect events. Wired-program tools are normally **external tools**: free standing devices that sense electronic signals in the circuitry of the measured system, and record them externally to the measured system. Counters can be used to implement timing meters and counting meters. Logic gates can be used to combine and sequence signals. Comparators and sequence detectors can be used to trigger recording devices, to produce event traces. The results of the event filtering can be displayed, or saved in a data base for later analysis.

In **Stored-program hardware tools** the logic plug-board is replaced by a computer-controlled event-filter. Filtering functions are set up by software, not by manual insertion and removal of modules in a plug-board. The logic-state analyser, is among the latest development in stored-program tools. The ability to control filtering with software gives stored-program tools greater flexibility, and if an interactive user interface is provided, greater ease of use than wired tools. Some early tools, however, did not have the resolution of wired tools.

**Characteristics of hardware tools**

* Hardware tools cause little, or no, interference to the system being measured, and thus, they can be used for long periods.
* They have high resolution - often greater than the clock frequency of the system under study. Simultaneous measurement of overlapped activities is possible.
* Some hardware related activities are accessible to hardware tools but not to software tools - for example data transfer in a buffered peripheral.
* Hardware monitors can be used on any system, provided the relevant signals are accessible. Software related events can only be sensed when they are accompanied by an instruction at a known address.
* The state of a memory location can only be monitored during read or write operations to that location.
* Due to their high resolution and low interference, hardware tools provide very precise readings.
* Attaching hardware probes to a computer makes maintenance men nervous. Poorly installed probes can load signals, hence impacting upon performance.
* Probe attachment takes time and skill. Improperly attached probes produce misleading information. A means of verifying that the readings are correct is essential.
* Hardware tools can be used to monitor the failure and restarting of the target system.

**Software Tools and Techniques**

Software tools consist of instructions added to the target process to gather data related to its performance. Generally, no additional hardware is required. The instructions added to the code (called software sensors or software probes) collect the data, reduce the data, store it in internal buffers, and transfer the buffer contents to backup store. Analysis and display of results are usually carried out at a later stage. The most common use of software tools is in the generation of system accounting logs. These logs are used by managers for capacity planning, and in the creation of customer accounts. The simplest log consists of a sequence of messages that indicate the start and termi­nation of activities, the time of day, and the job name to which the activity is related. **More sophisticated logs include the amount of CPU time, the amount of I/O time per device, and the memory usage of individual jobs**. An accounting program is then used to collect job related information for each job, and to produce customer accounts.

Analysis programs can produce a complete record of the day’s activity, and calculate system statistics (for example the utilization of various resources and the throughput for different job classes), from the logged data. Careful use of system logs can provide a lot of the measures needed for evaluation. However, these logs are normally designed for accounting purposes, often with extra information thrown in for interest, and thus, may not include some desirable measures. Another log, provided by some systems, is a report to the user, at termination of his job, on the real time and CPU time used by the job.

A useful debugging tool, found in some software tool kits, is a program-execution profiler. Prior to compilation, a preprocessor inserts checkpoints to detect variable accesses and flow-of-control branching. During execution, these checkpoints increase counters. After execution a profile, showing either the number of times each line of code was executed or the number of variable accesses, is displayed to the user. All software tools are **event driven**. Sampling tools execute in response to external events. When an event occurs, the state (often the operating-system tables) of the target process (system) is read and recorded. Collected data is analysed using statistical methods. Sampling techniques reduce the amount of data needed to estimate some quantities. In a sense, they are equivalent to taking the pulse of a system. Sampling can be done at periodic or random intervals. In time sampling, the event which initiates the sampling process is the termination of a specified time interval. In count sampling, counting meters initiate the sampling process, when a specified count is reached, for example every n disc accesses. The counting meter can be a hardware device, or a software device: a memory location that is incremented every time a procedure executes. Event detection is done by the sampling program periodically checking the value in the meter. When the meter reaches the desired value, the sampling program resets the meter and records the desired state information. Events can also be detected by the meter routine, which then calls the sampling program. In this case we have an internally-driven tool. The essence of sampling is that the measures are synchronized to the termination of a sampling period, not to changes in the internal state of the system.

**Internally-driven tools**, by contrast, execute in direct response to the occurrence of events within the system. Thus, they are synchronized to the internal state of the system. Internal events are all detected by the execution of a piece of software, called a checkpoint or software probe, inserted into the target process. In an interrupt-driven operating system, each invocation of a module of the operating system is caused by an interrupt or a trap (supervisor call or software interrupt). These interruptions occur in response to changes of system state, and cause changes in the system state. Thus, a logical place to put checkpoints is in these routines. This is a neat way of decomposing the system object. Monitors based upon checkpoints in interrupt routines are called **interrupt-intercept monitors**.

Checkpoints can be inserted at various points of interest in the target process, not only to detect the execution of an instruction, but also, to detect a data structure being updated or a variable being assigned a certain value. Thus, they can be used to get at software specific information; for example user name, disc-file name, variable contents, job class, etc. In order to use system resources, read system tables, or get around system security, a checkpoint may have to use a supervisor call. A checkpoint can either collect data itself, or call a measurement program for the same purpose. Having recorded the required data, the checkpoint returns to the calling process. The data is usually stored in a set of in memory registers. When these registers are full, or when the experiment is complete they are transferred to backup store.

A **fixed software-tool** is a permanent checkpoint which collects data every time that section of the target process is executed. Accounting routines use fixed tools. Software meters are usually fixed tools. A software timing-meter increments a memory location when the clock routine executes if a flag is set, for example user time is incremented while ever a user process is executing. One way of minimising the overhead associated with incrementing software meters is to implement the counting routines in firmware. Tools which can be enabled and disabled, or inserted and removed, at will are non-fixed tools. In some non-fixed tools the event to be detected can be modified, or the data variable to be recorded can be changed. These facilities enable the user to tailor the measurements to the experiment at hand, and eliminate the execution of unwanted checkpoints, reducing interference. Systems instrumented in this way either have a set of permanently installed tools which can be enabled by a monitoring process or have facilities for automatic insertion and removal of checkpoints.

Software tools have some advantages, relative to hardware tools, and some disadvantages: •

* Software tools interfere with the system. Checkpoints take time to execute and use up memory. If the software tool uses less than 5% of system resources, then the measurement accuracy is generally adequate. One method of reducing interference, where a writable control store is available, is to implement checkpoints as firmware routines callable as assembler instructions.
* Resolution is lower than for hardware tools. They are most suited to recording macroscopic, infrequent events.
* They can record events only in a sequential manner, and they stop the execution of the target process while the data is being recorded.
* Hardware related events can only be detected if they are accompanied by the execution of a program instruction of the updating of a fixed memory location.
* Also, peripheral devices can only be monitored through their communications with the central processing unit.
* Software monitors can only be used on the system they were designed for.
* Software related information; for example program name, variable contents, and dynamic data structures; can easily be sensed by software probes.
* The state of memory locations can be monitored at any time by software tools.
* Software tools can only provide rough timing measurements, depending upon the precision of the system clock.
* Insertion of faulty software probes into a system can cause program faults, and may even cause the system to crash.
* Software tools are usually easier to install than hardware tools, particularly for programmers, and may be more flexible. Also they generally cost less.
* Changes to an operating system can drastically affect the accuracy of software tools, requiring compensating modification to the software tools.
* Software tools can handle dynamic environments which create problems for hardware tools: relocation of code modules, virtual memory, recursion dynamic data structures, and interpretation of programs.
* To use a software tool, the system must be instrumental to the level that pauses in the execution of the target process, due to an external interrupt, etc., are detected.
* A hardware tool can pick this, simply by detecting references to address outside the program area.
* A major headache with software tools is verifying their accuracy.

**Hybrid Tools and Techniques**

Hybrid monitors attempt to take advantage of the complementary nature of hardware and software tools. In a typical hybrid monitor, software tools detect events within the system and write information relative to those events to a hardware interface. Data arriving at the hardware interface is recorded, together with other hardware signals, by an external hardware tool, where it is analysed and displayed. Not all hybrid tools fit this pattern, a wide range of variations is possible. Hybrid monitors have the potential of being able to measure all the information about a system, with the precision of a hardware tool, but with less interference than a purely software tool. In a sense, a hybrid monitor is an intelligent peripheral, or at minimum an intelligent alternative to a backup store.

The major design decision to be made when developing a hybrid monitor is: which section should measure what? If you do not have a well thought out philosophy of hybridisation, the result may turn out to be an ad hoc collection of dissimilar components rather than an integrated measurement system.

Some hybrid tools are simply a combination of components from existing hardware tools and software tools; others are designed with a specific hybridization goal in mind. Hybrid tools can be loosely classified according to the partitioning of functions between hardware and software. All tools aim to keep the interference of the software tool to a minimum. Some however, do this only during the execution of an instrumented process and use considerable target system resources to set up the target process.

A major consideration in the use of hybrid tools is the selection of sensors. Some of the criteria to be considered when selecting the sensors to be used are:

* Your formulation of measurement and your philosophy of hybridisation.
* The number and type of quantities to be measured.
* The sensors which can be used to measure each quantity.
* Of the sensors available, which one has the most advantages and the least disadvantages. • Is data reduction to be done at the sensor or in the external tool.
* Can the sensors be installed easily.

Hybrid tools add a new element to our basic measurement tool configuration: the interface between the software and the hardware tool. This interface can be as simple as a parallel output port, or as complex as a direct memory access channel. Some interfaces are passive, they present information to the hardware tool under control of the software tool; others are dynamic, the hardware tool can request information from the software tool. Some interfaces allow the hardware tool to pass information to the software tool, or to interrupt the target process. These features are useful for feedback control of operating systems, and in debugging environments, where the analyst may want to stop and restart the target process.