Estimating Program Run Time

Estimating Program Run Time

- A real-time program is defined as a program for which the correctness of operation depends on the logical results of the computation and the time at which the results are produced.
- In general, there are three types of programming : Sequential, multi-tasking and real time.
- Estimating program run time depends on following factors :

1. Source code : Source code that is carefully tuned and optimized takes less time to execute.
2. Compiler : It maps source level code into a machine level program.
3. Machine architecture : Executing program may require much interaction between the processor and the memory and I/O devices.
4. Operating system : OS determines such isssues as task scheduling and memory management. Both have major impact on memory management.

Analysis of Source Code

- Consider the following code :

```
a := b x c;
b - d + e;
d = e - f;
```

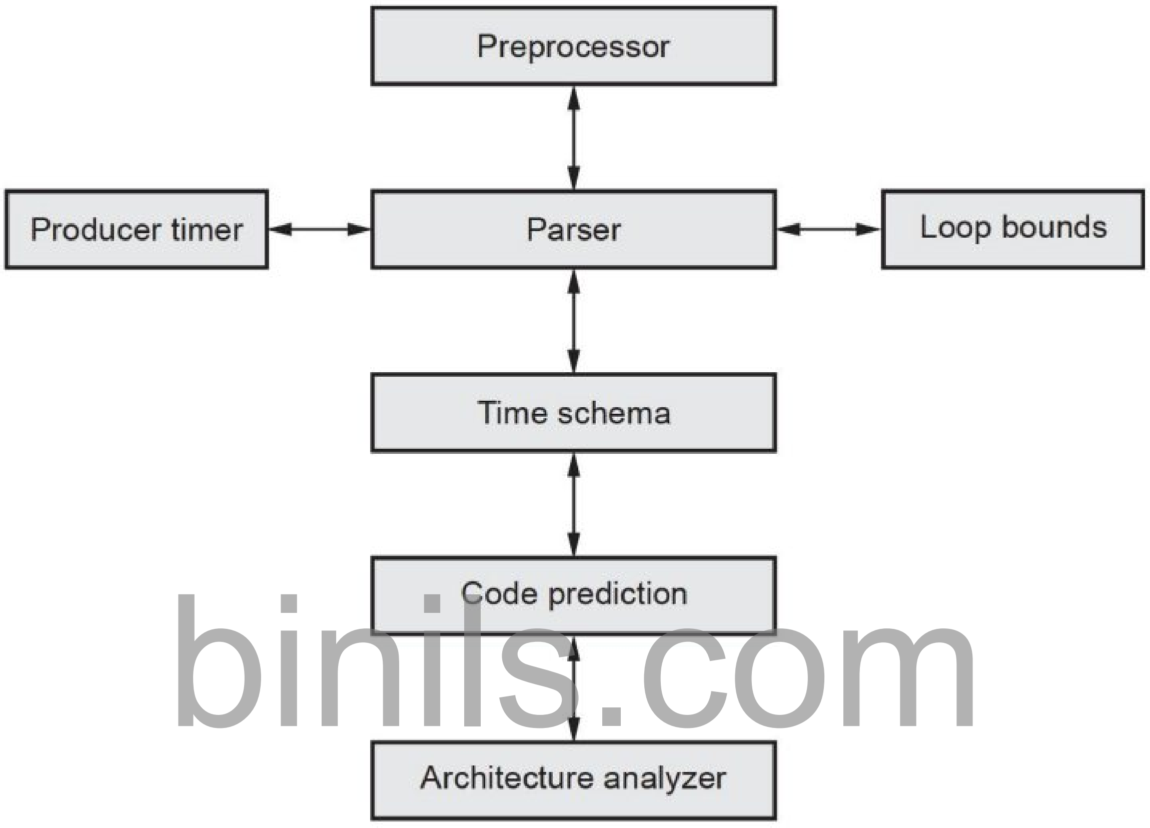
- This is straight line code. The total execution time is given by,

$$\sum_{i=1}^{3} T_{exec}(Li)$$

Where Texec(Li) is the time needed to execute Li.

- Execution time analysis is any structured method or tool applied to the problem of obtaining information about the execution time of a program or parts of a program.

- The fundamental problem that a timing analysis has to deal with is the following : The execution time of a typical program (or other relevant piece of code) is not a fixed constant, but rather varies with different probability of occurrence across a range of times.

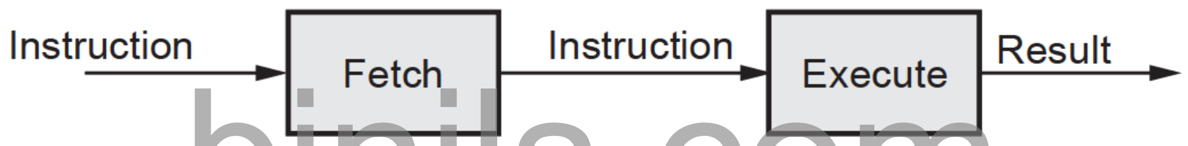


Schematic of timing estimation system

- Preprocessor produces compiled assembly language code. Parser analyze input source program.
- Procedure timer maintains a table of procedures and their execution times.
- The loop bounds module obtaines bounds on the number of iterations for the various loops in the system.
- The time schema is independent of the system, it depends only on the lanugaue.
- Code prediction module does this by using the code generated by the preprocessor and using the architecure analyzer to include the influence of the architecture.

Accounting for Pipeling

- Pipelining is the ability to overlap execution of different instructions at the same time.
- In a 2 - stage pipeline, you break down a task into two sub-tasks and execute them in pipeline. Lets say each stage takes 1 cycle to complete.
- That means in a 2-stage pipeline, each task will take 2 cycles to complete (known as latency).
- An instruction has a number of stages. The various stages can be worked on simultanously through various blocks of production. This is a pipeline. This process is also referred as instruction pipeling.
- The pipeline of two independent stages : Fetch instruction and execusion instruction. The first stage fetches an instruction and buffers it.



Pipeline of two independent stages

- While the second stage is executing the instruction, the first stage takes advantage of any unused memory cycles to fetch and buffer the next instruction. This process will speed up instruction execution.
- Several factors serve to limit the pipeline performance. If the six stage are not of equal duration, there will be some waiting involved at various pipeline stage.
- Another difficulty is the condition branch instruction or the unpredictable event is an interrupt. Other problem arise that the memory conflicts could occur. So the system must contain logic to account for the type of conflict.

Task Assignment and Scheduling

Scheduling real time tasks on distributed and multiprocessor systems consists of two subproblems :

1. Task allocation to the processor
   a. The task assignment problem is concerned with how to partition a set of tasks and then how to assign these tasks to processors -- task assignment can be : **1) Static or 2) Dynamic.**
   b. In the static allocation scheme, the allocation of tasks to nodes is permanent and does not change with time.
   c. In the dynamic task assignment, tasks are assigned to the nodes as they arise, different instances of tasks may be allocated to different nodes.
2. Scheduling of tasks on the individual processors : Uniprocessor scheduling algorithms can be used for the task set allocated to a particular processor.

Static allocation algorithms

The tasks are pre-allocated to processors.

No overhead incurs during run time since tasks are permanently assigned to processors at the system initialization time.

1. Utilization Balancing Algorithm
2. Next-Fit Algorithm for RMA
3. Bin Packing Algorithm for EDF

Dynamic allocation algorithms

In many applications tasks arrive sporadically at different nodes.

The tasks are assigned to processor as and when they arise.

The dynamic approach incurs high rim time overhead since the allocator component running at every node needs to keep track of the instantaneous load position at every other node.

1. Focussed Addressing and Binding (FAB)
2. The Buddy Strategy Algorithm

Utilization-Balancing Algorithm

This algorithm attempts to balance processor utilization, and proceeds by allocating the tasks one by one and selecting the least utilized processor.

Objective to balance processor utilization, and proceeds by allocating the tasks one by one and selecting the least utilized processor.

Maintains the tasks in a queue in increasing order of their utilizations. It removes tasks one by one from the head of the queue and allocates them to the least utilized processor each time.

The objective of selecting the least utilized processor is to balance the utilization of different processors.

$$\frac{\sum_{i=1}^{p}(u_i^B)^2}{\sum_{i=1}^{p}(u_i^*)^2} \leq \frac{9}{8}$$

Where u_i^* = P_i's utilization under an optimal algorithm that minimizes

$\sum$utilization ^{2}

u_i^B = P_i's utilization under best-fit algorithm

Next-Fit Algorithm for RM-Scheduling

This is a utilization-based allocation heuristic. The task set has the same properties as for the RM uniprocessor scheduling algorithm.

M is picked by user.

Corresponding to each task class is a set of processors that is only allocated to tasks of that class.

It is possible to show that this approach uses no more than N times the minimum possible number of processors.

There are m classes of tasks such that, each class of tasks are assigned to a corresponding set of processors.

T_i belongs to class $j < m$ if

$$2^{\frac{1}{1+j}} - 1 < \frac{e_i}{p_i} \leq 2^{\frac{i}{j}} - 1$$

T_i belongs to class m otherwise.

Bin-Packing Assignment for EDF

Same assumptions on tasks and processors as Next-fit algorithm.

Problem : Schedule a set of periodic independent preemptible tasks on a multiprocessor system consisting of identical processors.

The task deadlines equal their periods and tasks require no other resources.

Solution : EDF - scheduling on a processor and task set is EDF - schedulable if $U \leq 1$

Assign tasks such that $U \leq 1$ for all processors.

The problem reduces to making task assignments to processors with the property that the sum of the utilizations of the tasks assigned to a processor does not exceed one.

Focused Addressing and Bidding (FAB) Algorithm

It uses dynamic allocations.

FAB is a simple algorithm that can be used as an online procedure for task sets consisting of both critical and non-critical real-time tasks.

Critical tasks must have sufficient time reserved for them so that they continue to execute successfully, even if they need their worst? Case execution time.

The non-critical tasks are either processed or not, depending on the system's ability to do so.

The guarantee can be based on the expected run time of the task rather than the worst-case run time (noncritical task).

THE UNDERLYING SYSTEM MODE IS : When a noncritical task arrives at processor $p_{i,}$ the processor checks to see if it has the resources and time to execute the task without missing any deadlines of the critical tasks or the previously guaranteed noncritical tasks — if yes, p_i accepts this new noncritical task and adds it to its list of tasks to be executed and reserves time for it.

The FAB ALGORITHM IS USED WHEN pi determines that it does not have the resources or time to execute the task in this case, it tries to ship that task out to some other processor in the system.

Every processor maintains two tables called : Status table and system load table

1. **STATUS TABLE** indicates which tasks have been already committed to rim including the set of critical tasks (which were preassigned statically) and any additional noncritical tasks that have been accepted, execution time and periods of the tasks.
2. **LOAD TABLE** contains the latest load information of all other processors of the system, the surplus computing capacity available at the different processors can be determined.

THE TIME AXIS is divided into windows, which are intervals of fixed duration, at the end of each window, each processor broadcasts to all other processors the fraction of computing power in the next window for which it has no committed tasks.

1. Every processor on receiving a broadcast from a node about the load position updates the system load table.
2. Since the system is distributed, this information may never be completely up to date.
3. As a result, when a task arrives at a node, the node first checks whether the task can be processed locally, if yes, it updates its status table if not, it looks for a processor to offload the task.

THE PROCESS OF OFF LOADING A TASK is based on the content of the system load table, an overloaded processor checks its surplus information and :

1. Selects a processor (called the focused processor) ps that is believed to be the most likely to be able to successfully execute that task by its deadline.
2. The system load table information might be out of date — the overloaded processor, as insurance against this, will decide to send Requests For Bids (RFB) to other lightly loaded processor in parallel with sending out the task to the focused processor ps, this is to gain time in case ps refuses the task.

The RFB contains the vital statistics of the task -- its expected execution time, any other resource requirements, its deadline, etc.

3. The RFB asks any processor that can successfully execute the task to send a bid to the focused processor ps stating how quickly it can process the task.
4. An RFB is only sent out if the sending processor ps estimates that there will be enough time for timely response to it.

Buddy Strategy

The buddy strategy tries to solve the same problem as the FAB algorithm, soft real time tasks arrive at the various processors of a multiprocessor and, if an individual processor finds itself overloaded, it tries to off load some tasks onto less lightly loaded processors.

The buddy strategy differs from the FAB algorithm in the manner in which the target processors are found.

STRATEGY

1. Each processor has 3 thresholds of loading : Under Loaded (TU), fully loaded (TF), and over loaded (TV).
2. The loading is determined by the number of jobs awaiting service in the processor's queue. If the queue length is Q, the processor is said to be in :
   a. State U (underloaded) if $Q \leq TU$;
   b. State F (fully loaded) if $TF < Q \leq TV$;

c. State V (overloaded) if Q > TV;

3. If in state U a processor is in a position to execute tasks transferred from other processors, if in state V, it looks for other processors on which to offload some tasks, if in state F will neither accept nor offload tasks from/to other processors.
4. When a processor makes a transition out of or into state U, it broadcasts an announcement to this effect.

THE TRANSITION to/from state U is broadcasted to a limited subset of processors called processor's buddy set, each processor is aware of whether any member of its buddy set is in state U. If it is overloaded, it chooses an underloaded member (if any) in its buddy set on which to offload a task.

ISSUES RELATED TO THE BUDDY SET

1. How the buddy set is to be chosen in a multi hop network? If too large the state-change broadcast will heavily load the interconnection network. If too small the success of finding an available U state processor will diminish.
2. If a node is in the buddy set of many overloaded processors, and it delivers a state-change message to them saying that now is underloaded. This can result in each of the overloaded processors dumping their load on this one processor and make it overloaded.
3. THE CHOICE OF THE THRESHOLDS TU, TF, AND TV, in general, the greater the value of TV, the smaller the rate at which tasks are transferred from one node to another.

Assignment with Precedence Conditions

Algorithm that assigns and schedules tasks with precedence conditions and additional resource constraints, basic algorithm idea is to reduce communication costs by assigning(if possible) to the same processor tasks that heavily communicate with one another.

UNDERLYING TASK MODEL : Each task may be composed of one or more subtasks. The release time of each task and the worst-case execution time of each subtask are given.

The subtask communication pattern is represented by a task precedence graph. Ee are also given the volume of communication between tasks.

It is assumed that if subtask si sends output to s2, this is done at the end of si; Associated with each subtask is a Latest Finishing Time (LFT).

The algorithm is a trial-and-error process -- assign subtasks to the processors one by one in the order of their LFT values -- for same LFT the subtask with the greatest number of successor wins.

Check for feasibility after each assignment, if one assignment is not feasible try another one, etc.

A threshold policy kcis followed when to characterize the volume of communication between tasks. When subtasks communicate a lot, if possible, they are assigned to the same processor.

Reliability Evaluation

Reliability refers to the property that a system can run continuously without failure. In contrast to availability, reliability is defined interms of a time interval instead of an instant in time.

A highly reliable system is one that will most likely continue to work without interruption during a relatively long period of time. This is a subtle but important difference when compared to availability.

If a system goes down on average for one, seemingly random millisecond every hour, it has an availability of more than 99.9999 percent, but is still unreliable.

Two methods are used for finding device failure rates : Collecting field data or life cycle testing in the laboratory.

The most common accelerate is temperature : The higher the temperature the greater the failure rate. The acceleration factor is given by the following equation,

$$R(T) = Ae^{-E_a/kT}$$

Where A is a constant, E_a is the activation energy and depends largely on the logic family used, k is the Boltzmann constant.

To measure how quickly an error can propagate, we use fault injection. This is best done on a prototype. Special-purpose hardware is used to simulate a fault on a selected line. The status of related lines is monitored using logic analyzers to determine how far the error propagates and how quickly. If a prototype is not available, a software simulation can be substituted.

Clock Synchronization

Clock C_i is a mapping

C_i : Real time → Clock time

At real time t, C_i (t) is the time told by clock C_i. The inverse function C_i (t) is the real time at which clock C_i tells time T.

Clock drife rate is the rate at which the clock can gain or lose time.

There are two reasons :

1. How the clock rate of computer is determined? The speed of the computer is a function of the clock rate. The clock period is choosen to be just long enough for signal propogation along the critical path of a computer circuit.
2. Some synchronization algorithm adjust the clock simuntaneously.

If a synchronization algorithm is employed in the system to compensate the time error, two main sources of error remains; the information about the time of a clock degrades due both to the drifts of the clocks (as before) and to the uncertainty of the delay that the messages carrying the time take to travel along the distributed network.

The drift of the clock usually is more influent than the message delays in the case the time-carrying communication is infrequent; with the decrease of the frequency of communication, the uncertainty due to the clock drift increases, while the uncertainty due to the message delays remains constant.

A trivial solution may be the decrease of the synchronization interval, but any clock synchronization schemes require the time information obtained through the communication to be processed.

Fault Tolerance Techniques

Fault - tolerance is defined informally as the ability of a system to deliver the expected service even in the presence of faults.

A common misconception about real- time computing is that fault-tolerance is orthogonal to real- time requirements. It is often assumed that the availability and reliability requirements of a system can be addressed independent of its timing constraints.

A real-time system may fail to function correctly either because of errors in its hardware and/ or software or because of not responding in time to meet the timing requirements that are usually imposed by its "environment."

Hardware fault is some physical defects that can cause a components to malfunction. Software fault is a bug that can cause a program to fail for a given set of inputs.

An error is a manifestation of a fault. The fault latency is the duration between the onset of a fault and its manifestation as an error.

An error latency is the duration between when an error is produced and when it is either recognized as an error or causes the failure of the system.

Error recovery is the process by which the system attempts to recover from the effects of an error.

Recovery from an error is fundamental to fault tolerance.

Two main forms of recovery :

1. Forward error recovery
2. Backward error recovery

Forward error recovery

Forward recovery, attempt to bring system to a new stable state from which it is possible to proceed (applied in situations where the nature if errors are known and a reset can be applied).

Forward error recovery continues from an erroneous state by making selective corrections to the system state.

This include making safe the controlled environment which may be hazardous or damaged because of the failure.

It is system specific and depends on accurate predictions of the location and cause of errors (i.e., damage assessment).

Examples : Redundant pointers in data structures and the use of self-correcting codes such as Hamming Codes.

Advantage forward - error recovery :

1. Less overhead

Disadvantages of forward recovery :

2. In order to work, all potential errors need to be accounted for up-front.
3. Limited use, i.e. only when impact of faults understood.
4. Cannot be used as general mechanism for error recovery.
5. Design specifically for a particular system.

Backward recovery :

Most extensively used in distributed systems and generally safest. It can be incorporated into middleware layers.

Backward recovery is complicated in the case of process, machine or network failure but no guarantee that same fault may occur again.

It can not be applied to irreversible (non-idempotent) operations, e.g. ATM withdrawal.

Advantage backward - error recovery :

1. Simple to implement
2. Can be used as general recovery mechanism
3. Capable of providing recovery from arbitrary damage.

Disadvantage of backward recovery :

1. Check pointing can be very expensive - especially when errors are very rare.

2. Performance penalty
3. No guarantee that fault does not occur again
4. Some components cannot be recovered

Causes of Failuer

There are three causes of failuer,

1. Errors in the specification or design
2. Defecuts in the components
3. Environmental effects

Mistake in the specification and design are very difficult to guard against. Many hardware failuers and all software failuers occurs such mistake.

If the specification is wrong, everything that processeds fro it, design and implementation, likely to be unsatisfactory.

Fault Types

Fault are classified as temporal behaviours and output behaviours.

1. Temporal behaviours classification

- **Fault are of three types :** Permanent, intermittent and transient.
- **Transient faults :** These occur once and then disappear. For example, a network message transmission times out but works fine when attempted a second time.
- **Intermittent faults :** These are the most annoying of component faults. This fault is characterized by a fault occurring, then vanishing again, then occurring. An example of this kind of fault is a loose connection.
- **Permanent faults :** This fault is persistent : It continues to exist until the faulty component is repaired or replaced. Examples of this fault are disk head crashes, software bugs, and burnt-out hardware.

2. Output behavioure classification

- **Malicious faults :** Inconsistent output.
- **Nonmalicious fault :** Consistent output errors.

- **Fail stop :** Responds to up to a certain maximum numbers of failuers by simply stoping, rather than putting out incorrect outputs. The component simply stops working. For instance, a hard disk which refuses to read or write.
- **Fail safe :** Its failuer mode is biased so that the application process does not suffer catastrophe upon failuers. A component under too much load is likely to fail. A fail safe system, on detecting a large amount of load, processes each request slower to avoid failure.

3. **Independence and correlation**

- Components failures may be independent and correlated.
- **Independent :** A failure is said to be independent it it does not directly or indirectly cause another failure.
- **Correlated :** If the failure is said to be correlated if they are related in some way.