CHAPTER 3

PROPOSED BLACK BOX ANALYTICAL MODEL

3.1 Introduction

In the previous chapter the different aspects for modeling Web applications by partitioning them into clusters, identifying FSMs for each cluster, combining these to represent the application itself as FSM, and the patterns in the state transitions of the FSMs, were identified from the literature and the groundwork for modeling laid. An elegant model based on '*thread-per-request*' architecture of server, to represent a variety of Internet applications and understand the nuances in implementation is presented in this chapter.

3.2 Modeling Workload

Workload characterization is a modeling process to reproduce the user access patterns to Web sites. In the study of workload characterization of traditional Web information servers, a request is the basic unit for analysis. Workload to a Web Server is viewed as a stream of requests. The service request is taken as a stationary random process confirming to normal distribution. The proposed service system considers a discrete random arrival pattern. Each arrival is independent of its previous arrivals. An arrival set is characterized as a set of requests, represented as $A\{$*Time of arrival of requests*(T_i), *Number of request arrivals*(Q_i), *Number of requests of each service category*($Q_{c1}, Q_{c2} \ldots Q_{cn}$) $\}$. Each request in the arrival set is represented as $R\{$*ArrivalTime*(A_i),*ServiceTime*(S_i), *Reward*(R_i),*Category*(C_i),*UserId*(U_i)$\}$. The arrival capacity is limited only by the system capacity (i.e., maximum number of threads supported by the server) and this is the maximum number of concurrent requests the server is capable of handling. The pseudocode for modeling arrival pattern is specified in Algorithm 3.1.

3.2.1 Modeling Peak load

Ideally a peak load can be looked upon as an impulse load of user requests superposed on the regular load (of stationary distribution).

A peak load impulse should conform to the following:

1. It should occur randomly.

2. Its magnitude should be at least one order larger than the regular average load.
3. The execution strain on the system due to it should be conspicuously higher than that of the regular load. In other words the impulse value should be at least one order larger than the possible maximum value of $\sum_{i-S}^{i-1} Q_i$ where Q_i is the number of service requests in T_i (i.e., i^{th} timeslot).

The model can be modified to study services with any surge type peak load superposed on regular loads. Service under these conditions with a large number of users seeking concurrent service can be accomplished.

Algorithm 3.1 Arrival

```
begin
2^n balls are to be assigned three indices each
k = 0
while (– (n – 1) + 2k  ≤ (n – 1))
Assign the first index to the next (n k) balls as μ – (n – 1) + 2
 k++
Assign third index serially from 1 to 2^n
R = 2^n
i = 0
      while (R)
          r = rand() % R
          if(r is not already assigned as second index)
             Assign it as second index for ball with i as second  index
             Break
          end if
       end while
i++
R--
end while
#Basket is ready
#Random selection of arrivals
R = rand() % 2^n
Pick the ball from the basket with the second index as r
The first index of this ball is the number of arrivals
End
```

3.2.1.1 Modeling Multiclass User Arrivals in Workload

An arriving request may belong to any service category that determines the QoS to be offered. The model, has room for calibrated services, so that, service rendered will be commensurate to the user service category. The model can generate arrival pattern of requests, in different timeslots, for each service category of user, their distribution being random with equal probability for each value. Algorithm 3.2 specifies the pseudocode for generating number of requests for each service category.

Algorithm 3.2 Multiclass–arrival

begin
Get the total number of arrivals for the timeslot using Algorithm 3.1
Assigning *r* arrivals randomly to the n service categories as *a*[1], *a*[2], .. *a*[n] with *a*[1] + *a*[2] +..+ *a*[n] = *r*

a[1] = rand()% *r*
.
.
.
a[n] = rand() % (*r* $-\sum_{i=1}^{n-1} a[n]$)
end

To make realistic estimations, the length of the experiments must be long enough (i.e., peak loads that span over a lengthy period of time). The network attributes like *source IP*, *Destination IP*, *and bytes transferred* do not play any role in the design of the scheme; hence their logs are omitted.

3.3 Modeling Application Characteristics

3.3.1 Service Architecture

Many researchers and practitioners have been trying to find an effective way to model Web applications. FSM based modeling has a long history in the design of Web applications [2]. A FSM is used to simulate processing in a server. In this study, we assume that each Web request comes with the following information:

- Service Category.
- Requested data.
- Service time required.
- Arrival time.

A sequential set of FSM states comprises the service architecture.

3.3.2 Modeling User Behavior

Recommender systems based on historic profiles of users, can offer enhanced quality of service to preferred user service category. In experimentation, initial one-third of the recorded log was used for training the system (collecting data for providing service differentiation) and the remaining were used for testing purpose (offering service differentiation). The pseudocode for providing service differentiation based on user behavior (number of visits to Website) is specified in Algorithm 3.3.

Algorithm 3.3 Service Differentiation

```
begin
# Number of hits of users was used to provide differentiated service
Let hit=0
Create a lookup table with desired user mix (e.g. 30% of known
users/registered users and 70% of unknown/new users)
Generate the arrival pattern of requests using Algorithm 3.1
for ( each request in the arrival lot )
    begin
        if (free thread is available and thread not assigned to the
          request)
           begin
             Accept the request for processing
             Get user characteristics from lookup table
             Assign request characteristics
             Process request and update the user_profile
             Update hit=hit+1
           end
        else
          begin
            Select requests with user_id's having hit count set below a
            defined threshold value and holding threads
            Terminate the selected requests and add the obtained
            threads from the requests to the Free pool
          end
         end if
end for
end
```

3.3.3 Modeling Session

A set of successive requests submitted by a user constitutes a session. Typical interaction of users with Internet Service is organized into sessions, and a sequence of related requests, which together achieve a higher level user goal. An example of such interaction is an on-line shopping scenario for an E-Commerce Website which involves multiple requests that:

1. Search for particular products.
2. Retrieve information about a specific item (e.g., quantity and price).
3. Add it to the shopping cart.
4. Initiate the check-out process.
5. Finally commit the order.

The duration of each session is different as individual users make their own decisions on how they should navigate in the application. **'BBAM-I'** – being explained in the sequel – is a basic scheme to model and study such Internet Services. In **'BBAM-I'** a session is characterized with the following two attributes:

Session Length: The number of requests submitted by a user in a session is called the '*session length*'. The model supports unlimited session length (because once the user is admitted into the system, service is guaranteed until termination/completion of transaction).

Workload Mix: Workload mix defines the proportion of service categories in which the site is visited by users. The model uses the recorded patterns of user behaviors to re-categorize user.

3.4 Proposed Model ('BBAM-I')

The scheme here uses the traditional '*thread-per-request*' model used by servers (APACHE). The service architecture of **'BBAM-I'** is illustrated in Figure 3.1. An FSM is an abstract machine consisting of finite number of states, input actions, output actions, and a finite number of transitions between states. The interaction between the users and the states in the FSM can be modeled using UBMG.

Working: At the beginning of every time slot threads from requests that completed execution in final state of FSM are returned to the free pool of threads. The incoming requests are assigned threads for execution until request list is exhausted or threads in the thread pool are exhausted whichever is earlier. The scheme does not call for any detailed decision making for thread allocation. Once a thread is allocated to a request, it remains tied to the request until completion of execution. There can be time slots when threads are free and those when requests are dropped though both do not happen simultaneously. The pseudocode of **'BBAM-I'** is specified in Algorithm 3.4.

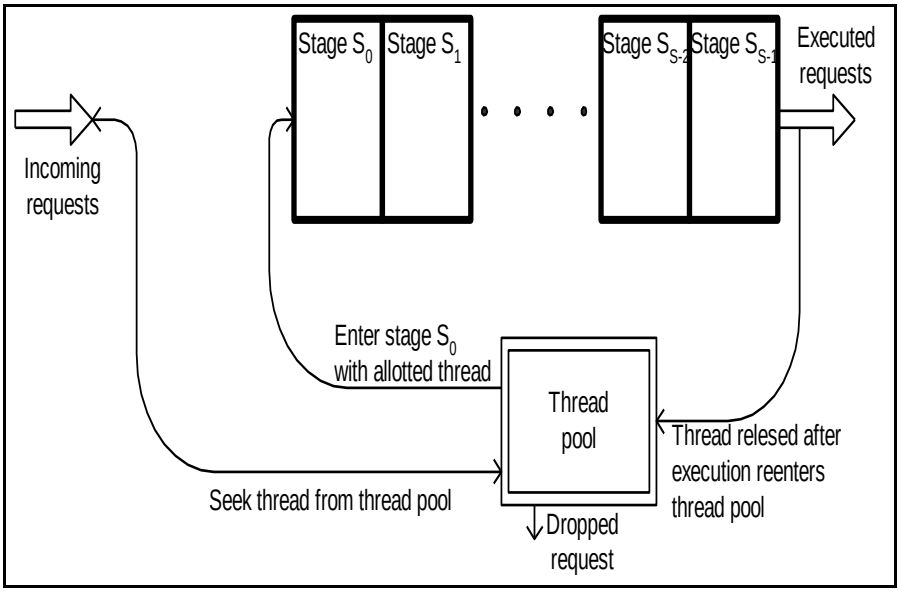


Figure 3.1: Service Architecture – **'BBAM-I'**

3.4.1 The Stage

The fundamental unit of processing represents a *stage*. A *stage* (Figure 3.1) is a self-contained application component. At the highest level of abstraction a *stage* represents the associated functionalities to achieve a sub-goal of the overall Web application processing with an integrated behavior model graph that represents the order in which these functions are invoked by users. At the next lower level of abstraction a *stage* represents all the associated Web pages used by the application to achieve the function and the associated behavior model that represents the transitions between Web pages. At the third lower level of abstraction a *stage* represents the steps in processing a Web page with the associated resource requirements. Figure 3.2 is a partial representation of such a '3-Tier' scheme. The UBMG at the first level can be used to generate virtual user's invocation of functionalities of the application and the UBMG at the next level can be used to generate the access patterns of users to invoke the Web pages associated with providing a specific functionality of the application. Conceptually such repeated splitting down of *stages* – in an 'inverted tree' form – can be carried through to the lowest grain level desired.

Algorithm 3.4 BBAM-I

```
begin
Generate request arrival pattern using algorithm 3.1 choosing
parameters μ and σ^2
Make resource commitments.{Maximum number of threads,
stages – based on FSM of Web application}
Register the metrics to be collected
for (each time slot)
    Generate the request lot for that time slot
    Free the threads in the final stage(As the requests would have
    got completed)
  while (more requests in the request lot)
        if (free thread is available)
        begin
           Allocate the thread to the request
           Collect logs of requests, stage and resource
        end
        else
           Drop the request
        end if
     end while
        Compute and record the registered SLA metrics {
        Throughput, Response time….}
  end for
end
```

3.5 Performance Evaluation

3.5.1 Workload

An extensive simulation study was carried out with the scheme. Its mean μ and variance σ^2 were varied over wide enough ranges to accommodate extremes of possible real life situations. As the deviation $((T/S) - \mu)$ increases the average drop rate reduces for all values of σ^2. Here T and S stand for maximum number of threads (system capacity) and maximum number of stages (number of states in FSM) respectively. Table 3.1 shows the variation of the probability of a request drop in a set of S successive timeslots at $((T/S) - \mu) < 2$ with σ^2; the same being negligible, the simulation need be carried out only for $((T/S) - \mu) < 2$.

Similarly for $((T/S) - \mu) > 2$, from the symmetry of the service distribution one can see that the un-serviced request rate also has an identical behavior. Combining these facts the simulation study was limited to the range $| (T/S) - \mu | < 2$ for all σ^2 values.

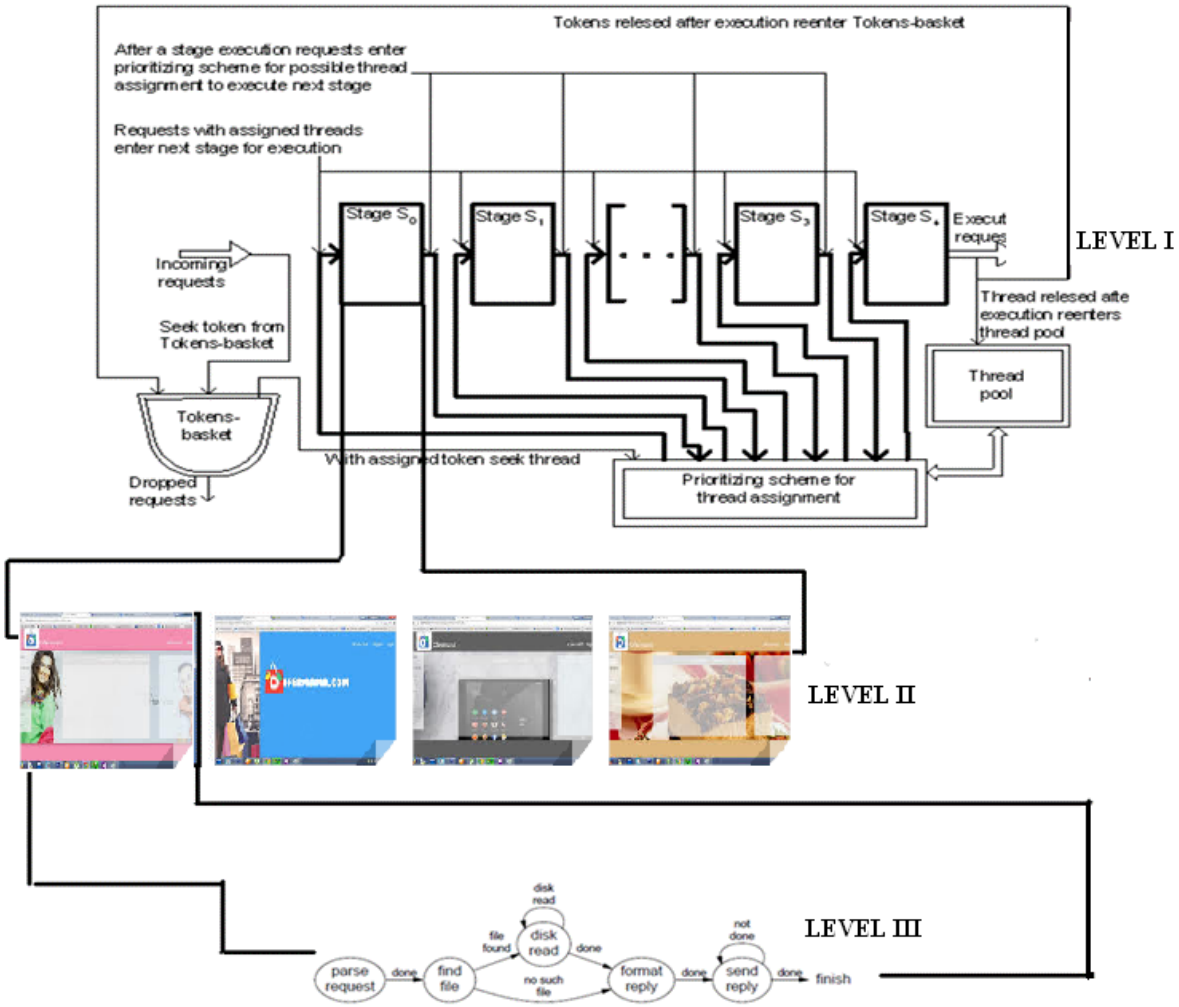


Figure 3.2: Levels of Abstraction of *Stage*

Table 3.1: Variation of Probability of Request Drop with σ^2

σ^2	2	3	4	5	6	7	8	9	10	11
P	0.003693	0.005909	0.022013	0.021387	0.044766	0.040345	0.067578	0.059460	0.088501	0.077501

The representative nature of a workload depends directly on the quality of its control parameters. In the experimental runs, several criteria (time varying nature, number of requests, and mixes of service categories) were used, to test the representative nature of the workload. The experimental measurement collection showed that the model clearly provided a concise representation of the workload characteristics.

3.5.2 Metric Evaluation

Request Drop (D_j): Consider the continuous operation of the system. t_j and x_i are the j^{th} time slot and the number of arrivals in the i^{th} time slot respectively. Total arrivals (A_T) in S successive time slots preceding are:

$$A_T = \sum_{j=0}^{S-1} x_{i-j} \tag{3.1}$$

The number of requests dropped in the j^{th} time slot D_j is given by $D_j = A_T - T$. For a given arrival pattern $\{\mu, \sigma^2\}$ one can get this quantity.

Response Time (R_T): With τ as the execution time of a request and w_i being the waiting time in the i^{th} stage before thread allotment, the response time is computed as:

$$R_T = \sum_{i=1}^{S-1} w_i + S\tau \tag{3.2}$$

Availability: Availability is the ratio of the number of requests serviced to the total number of requests arrived.

User Frustration: User frustration is modeled in **'BBAM-I'** as a measure of the number of times the user's request has been dropped based on history of visits. User frustration of a customer is the ratio of the number of requests of the customer dropped to the total number of requests made by that customer (hits).

3.5.3 Simulation Results

The simulation run duration was selected with the two following criteria:

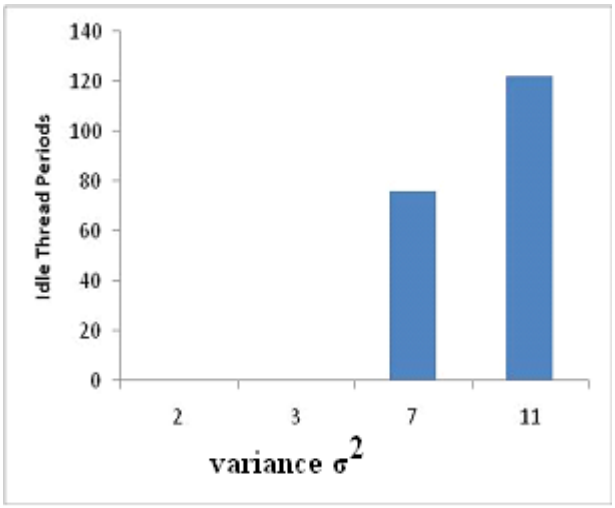
1. All possible service request values should appear enough number of times in the simulation to bring out all behavioral characteristics.

2. The transients associated with the initial part of the simulation, should be kept away in culling out the data representing regular steady operation. Further for each μ selected σ^2 was varied over the full range. With each of these sets random arrival patterns were generated for the study. Extensive simulation was carried out each time over a large range of time slots. The results have been consolidated and representative cases have been presented in Table 3.2.

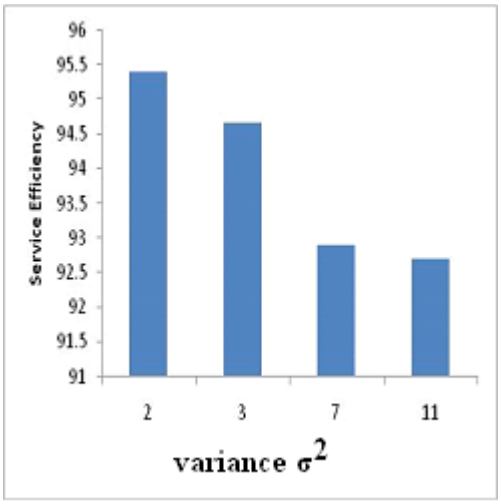
Table 3.2: Summary of Logs for S = 5, T = 100, μ = 22, and Varied σ^2 Values

Scheme – **BBAM-I**				
Number of threads(T) 100				
Number of Stages(S) 5				
Number of Timeslots of simulation 100				
Mean(μ) = 22				
Variance(σ^2)	**2**	**3**	**7**	**11**
Total Submitted requests(Total Arrivals)	1980	1986	2000	2000
Request in progress - (Will be realized in successive time slots)	100	100	100	100
Throughput	1889	1880	1858	1854
Idle thread periods	0	0	76	122
Service Efficiency (%)	95.404	94.6626	92.9	92.7
Number of Dropped Requests	101	106	152	156
System Utilization (%)	100	100	99.20	98.72

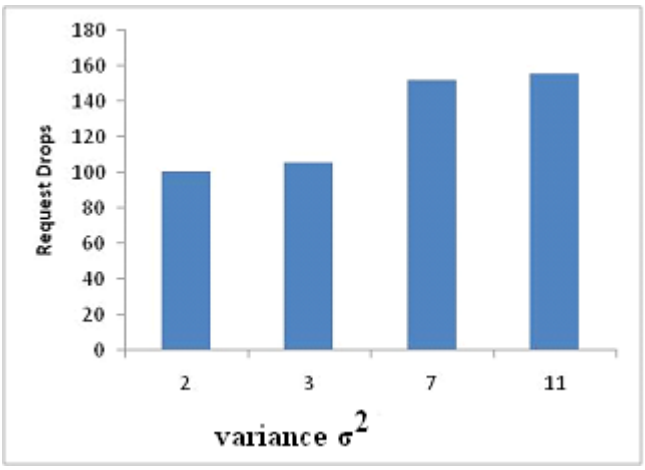
With the above procedure the system was able to provide same level of user satisfaction to all the accepted user requests. With service differentiation introduced in the scheme, to differentiate users based on their history of visits, the system was allowed to drop the user requests if number of hits were below a set threshold value. These free threads were used to accommodate new requests. Graphs in Figure 3.3 illustrate the measure of service level offered to users based on the collected profile. It was observed that, until the system was stable all the users were serviced. With increase on the load, users were denied access based on their previous history of visits. From the graph in Figure 3.4 it can be seen that user with ID=85 was dropped 8 times, and served 21 times out of the total visits of 29 times.



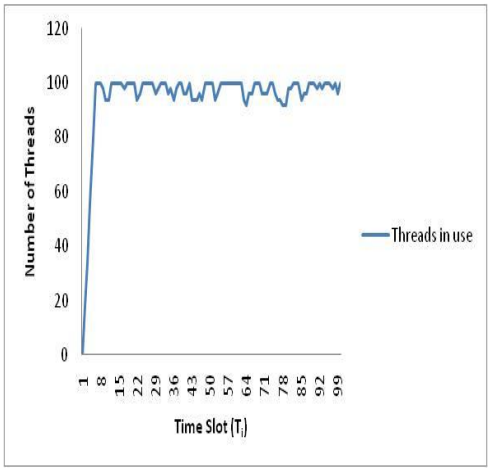
(a)



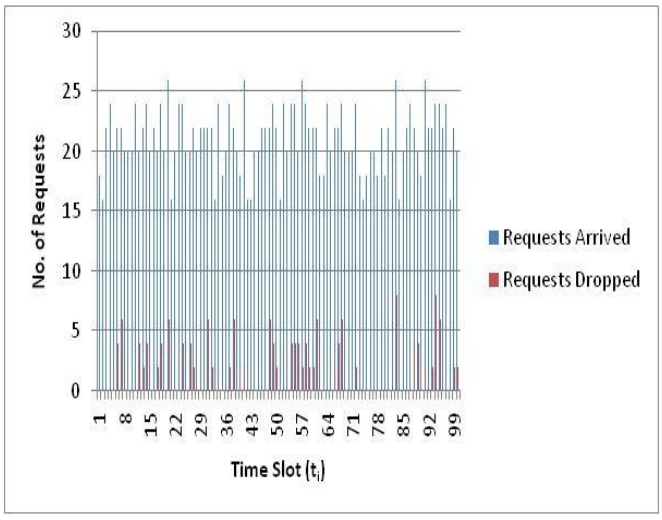
(b)



(c)



(d)



(e)

Figure 3.3: Performance details with simulations runs for 100 timeslots.

(a) Idle Thread Periods
(b) Service Efficiency
(c) Request Drops
(d) Resource Utilization
(e) Request Drops(Timeslotwise)

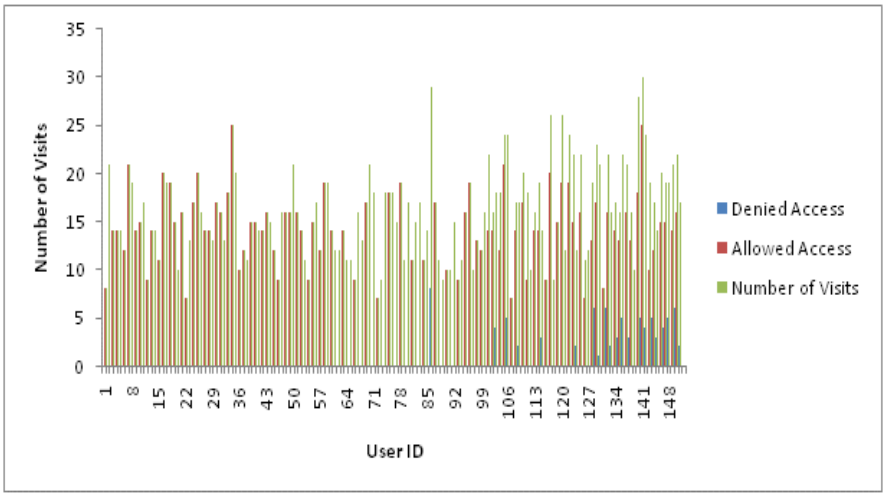


Figure 3.4: User Satisfaction Measured as Number of Allowed Accesses Vs. Denied Accesses

3.5.4 General Observations from the Logs

1. The maximum possible continuous service request execution rate is of $T/\tau S$ requests per ns.
2. If R is the average request rate $L=(T/\tau S) - R$ is the latency margin available as a cushion to meet peak / emergency loads.
3. Stable operation is possible only if $L \geq 0$.
4. With Q_i as the number of service requests in the i^{th} time slot, in general a service request at the i^{th} time slot can be denied service if $\sum_{i-S}^{i-1} Q_i > T$. Here $\sum_{i-S}^{i-1} Q_i$ is a function of the average arrival rate R, L, and the variance σ^2.
5. In turn the request denial rate is a direct function of σ^2 and an indirect function of L. Further the thread utilization T_u defined as (average number of threads utilized / T) approaches 100% as $L \rightarrow 0$ and also as $\sigma^2 \rightarrow 0$.
6. Service efficiency defined as (average number of requests serviced / average request rate) approaches 100% as long as $L > 0$ and as $\sigma^2 \rightarrow 0$.

3.5.5 Discussions

Ideally independent of the σ^2 value selected, the cumulative number of requests should linearly increase with mean value μ. As was explained in Section 3.5.4 as long as the margin $L \leq 0$ the throughput was able to cope up with the variations in the load but for $L > 0$ the throughput is not able to cope up with the demand; the variation of service efficiency and drop rates with σ^2 in the graph 3.3c brings this out. Further one can see that service efficiency is practically 100% for all $L \leq 0$; the deficiency here increases with σ^2 values (Figure 3.3b). In line with this the drop rate also increases

marginally as σ^2 increases and the disparity is more with increase in L. Similar observations hold good for system utilization also (idle thread periods). Graphs in Figures 3.3a, 3.3d, and 3.3e bring this out clearly. The graph in Figure 3.3e shows the access rates and drop rates of users due to non availability of threads.

3.5.5.1 Peak Load

'BBAM-I' does not address peak loads on servers as once the number of threads in the server exhausts, further requests that arrive in that time slot are dropped. Further acceptance of requests is done only when one or more threads in the system become free.

3.6 Summary of Findings

In a traditional '*thread-per-connection*' when all server threads are busy, the server stops accepting new connections; this is the type of overload protection used by APACHE Web Server [66]. The serious problems with this approach are that, the performance in terms of service availability is based on the number of threads rather than on the user load / length of time. Not accepting new connections, gives the user no indication that the site is overloaded. Finally, this approach is extremely unfair to clients, as some clients will be able to establish connections quickly, while others will experience large back off delays i.e. the scheme is not efficient at times of increased workloads. This scheme is useful only for applications where resource guaranteeing is more important than serving higher number of concurrent users (e.g., a multimedia streaming application with a periodic time deadline). That is, the scheme is useful for an Internet Service where, the focus is on individual requests, for which it is permissible (and often desirable) to meet statistical performance targets over a large number of requests.

3.7 Conclusions

The basic model for studying the performance of Web applications based on *'thread-per-request*' service architecture has been presented in this chapter. The model is general enough to capture workloads imposed on a service, to measure, and to update model parameters. The implications of this architectural model at times of overload are as follows:

1. Thread pool exhaustion lead to slow or unresponsive server and service becomes unavailable when it is needed at most.
2. Users who managed to get connected to the server will enjoy high quality of service and on the other hand other users waiting to get a connection will be starved.
3. Request rejections lead to user frustration which may have different impact on different job profiles for e.g. in E-Commerce, it takes dissatisfied customers only seconds to leave a site and take their business to a competitor.

It is to be emphasized here that, the scheme is too simplistic to be adopted by reward driven Internet Services. However, the scheme was selected to understand the nuances of an implementation and make a case for a sophisticated one. Still, the model could be useful with new Internet Services that are yet to become popular.