I wrote a console program for answering for the interview question, which imitated the ticket selling in a typical stressful ticket flood scenario.

It is implemented using a greed algorithm to allocate the ticket for incoming requests, which is explained as below:

If a new ticket request is arriving, the allocation process takes the ticket start station, and finds among the available seats for the one that has the biggest stop station upon which the tickets were sold. And sell the seat to satisfy this request.

The vacancy rate is computed to make the allocation algorithm’s performance known.

In real situation, such as web-based application, the concurrency bottleneck is attacked by both software architecting and the deploy structure.

In the application layer, we can divide this as two steps solution. Firstly capture all ticket requests, which are the web requests. Then we have back end program to process the requests, there will be time overlap between these two steps as it’s concurrent in nature. We can spawn threads for ticket processing for the captured requests. We can use memory cache, and distributed database to accommodate the big data storage and retrieval.

We can deploy the application with software and hardware load balance. Though for real big data problems, we have to add more hardware actually. An optional deploy structure is a server farms with multiple state servers connecting to subsets of web servers with a connected star topology.

Attached below is a run for the imitation program.

