Performance

- There are two common metrics for performance:

1. **Items per unit time**:

- This might be transactions per second, jobs per hour, or some other combination of completed tasks and units of time.
- Essentially, this is a measure of bandwidth. It places the emphasis on the ability of the system to complete tasks rather than on the duration of each individual task.
- Many benchmarks are essentially a measure of bandwidth.
- If you examine the SPEC Java Application Server benchmark (SPEC jAppServer1), you'll find that final results are reported as transactions per second.
- Another example is the linpack benchmark used as a basis for the TOP5002 list of supercomputers. The metric that is used to form the TOP500 list is the peak number of floating-point operations per second.

2.Time per item.

- This is a measure of the time to complete a single task.
- It is basically a measure of latency or response time.
- Fewer benchmarks specifically target latency.
- The most obvious example of a latency-driven benchmark is the SPEC CPU benchmark suite, which has a speed metric as well as a rate metric.

Quality of service (QoS)

- Metric that they must meet. The QoS metric will specify the expectations of the users of the system as well as penalties if the system fails to meet these expectations. These are two examples of alternative metrics:
- Number of transactions of latency greater than some threshold. This will probably be set together with an expectation for the average transaction. It is quite possible to have a system that exceeds the criteria for both the number of transactions per second that it supports and the average response time for a transaction yet have that same system fail due to the criteria for the number of responses taking longer than the threshold.
- The amount of time that the system is unavailable, typically called downtime or availability. This could be specified as a percentage of the time that the system is expected to be up or as a number of minutes per year that the system is allowed to be down.

- Knowing the available time for an update might influence the following decisions:

    - How many threads should be used to process the update. A single thread may not be sufficient to complete the update within the time window. Using the data, it should be possible to estimate the number of threads that would be needed to complete the update within the time window. This will have ramifications for the design of the application, and it may even have ramifications for the method and format used to deliver the data to the application.

    - If the update has to be stored to disk, then the write bandwidth of the disk storage becomes a consideration. This may be used to determine the number of drives necessary to deliver that bandwidth, the use of solid-state drives, or the use of a dedicated storage appliance.

    - If the time it takes to handle the data, even with using multiple threads or multiple drives, exceeds the available time window, then the application might have to be structured so that the update can be completed in parallel with the application processing incoming transactions. Then the application can instantly switch between the old and new data. This kind of design might have some underlying complexities if there are pending transactions at the time of the swap. These transactions would need to either complete using the older data or be restarted to use the latest version.