Predictive tuning

Predictive tuning is the process of tuning the system for its intended deployment environment, rather than for its measured (test) environment.

Much effort is often expended in re-creating components of the production environment to test for performance constraints in a system. However, this typically involves deploying the application in some sort of pilot, long after application changes can easily be implemented. Hence, these tests serve to define the system's limitations rather than enable its expansion.

Predictive tuning is based on a lab-friendly testing solution that analyzes the performance of the integrated system—in particular the performance relationships between system architecture, application logic, network characteristics, and server processing speed—in software.

Predictive tuning classifies the transaction as server (processing)-bound, network bandwidth-bound, or network latency-bound. It then quantifies the sensitivity of each application thread to the predicted performance constraint.

The result is a clear understanding of the performance effect contributed by each component of the integrated system—network, nodes, and application—and how these relate to each other. This method provides organizations the opportunity to correct problems before deployment by defining actionable and measurable performance improvement solutions.

The elements of predictive tuning

Effective predictive tuning requires some basic information related to the application to be analyzed.

Test environment

The test environment needs to include a client machine, the servers supporting the application, a network connecting these together, and the distributed application to install on these. In other words, you must be able to execute the transaction across a local network from a client machine. DNA captures the transaction's data packets from the network at each tier to perform its analysis. These predictive tuning steps can be similarly applied for applications deployed in a distributed WAN environment.

Business performance goal

To determine what, if anything, can or should be done to tune the system, it is valuable to understand what the end-users of the system expect in terms of response time. This helps identify what efforts are needed to meet these goals. Alternatively, predictive tuning can be used to evaluate the system's performance efficiency, essentially as a means to validate acceptable production behavior.

Targeted deployment environment

Predictive tuning requires a simple definition of the intended deployment environment. This involves understanding where the servers are located and basic abstractions of the network paths to be used to connect the users to these servers. Where the network does not yet exist, or where server locations have not yet been defined, predictive tuning can be used to determine appropriate network infrastructure requirements and ideal server locations.

The process of predictive tuning

DNA's capture agents provide a simple means of capturing transaction traces from the test network. You will want a separate capture of each transaction to be reviewed for tuning.

DNA uses readily available definitions of bandwidth (network speed), latency (network delay) and load (network utilization) as primary inputs to its response time prediction algorithms. The prediction output quantifies each transaction's performance in terms of its sensitivity to the core elements of the integrated system: server processing, client processing, network bandwidth, network latency, and network load. DNA's response analysis and prediction can then be used to prioritize for review the application threads most sensitive to this environment. Focus can then be applied to tuning those threads that will consume the greatest portions of the response time budget, or to tuning the part of the physical system that will become the bottleneck.

DNA's Response Time Predictor interface allows you to test various tuning options to validate their effect and estimate the improvement to be gained.

Predictive tuning examines the most significant bottlenecks by analyzing the interaction between the application logic and the physical system components—processing nodes and transmission media. Therefore, for each bottleneck, there will be both logical application and physical system tuning opportunities. These bottlenecks and predictive tuning options are listed in the table below. You will want to evaluate these options based on their costs (such as, development time, hardware costs, or recurring bandwidth expense), and the predicted performance improvement.

Processing delays

- Identify the threads that incur the most significant processing delays and investigate application tuning options for these threads.
- Evaluate increasing node processing power.

Bandwidth delays

- Identify the threads that transmit the largest number of bytes and investigate reducing the amount of data sent (by compression, caching, or other application efficiencies).
- Evaluate increasing network bandwidth.

Latency delays

- Identify the threads that incur the largest number of application turns and investigate reducing the number of network round-trips performed by the application logic.
- Evaluate reducing network latency (by re-routing traffic, relocating servers, or using terminal server technology).

Congestion delays

- Identify the threads that transmit the largest number of bytes and investigate reducing the amount of data sent (by compression, caching, or other application efficiencies).
- Evaluate reducing congestion (or the effects of congestion) by packet shaping, QoS policies (such as priority queuing), reducing or re-routing other traffic, or adding bandwidth.

TCP delays

- Identify the threads that transmit payload greater than the receiver’s TCP window and investigate reducing the amount of data sent (by compression, caching, or other application efficiencies).
- Reduce the number of TCP connections required by the application (by using HTTP persistent connections or increasing the TCP connection time-out).
- Increase the receiver’s TCP window setting (if there are threads that transmit payload greater than the window setting).