

# Mixed Workload Design Priority Guidelines

## Summary

This white paper briefly describes the high level design for the Mixed Workload Design and presents some general guidelines to control mixed workload applications using priority settings.

## MWE Basic Design

The HP NonStop Kernel operating system uses a priority-based design to control process dispatch. The highest priority executable process on the ready-list is dispatched.

Applications rely on a variety of system services, including communication, security, transaction, and data services. These services are typically provided using a client-server model, using non-stop process pairs. The heart of the database engine is the data access manager, more commonly referred to as DP2.

By definition, DP2 runs at system priority (220), which ensures that the operating system and transaction services function properly.

### **Definition: Priority Inversion**

**When a service process executes at system priority, a low priority transaction can utilize high priority services.**

**Low priority workload requests for services can delay service for higher priority workloads. This behavior is termed “priority inversion”.**

**To prevent priority inversion, the service process must not service lower priority requests while higher priority workloads are waiting for any resources, AND must preempt active service when higher priority workloads require new resources.**

The use of message priority by the high priority system service (DP2) to prevent priority inversion includes both CPU and I/O resource contention. The methods used to resolve this contention include the following:

1. CPU context switch, which compares the priority of the currently active request with the priority of processes on the ready-list. This is used to resolve conflicts for the CPU resource.
2. Request context switch, which compares the priority of the currently active request with the priority of the message queue. This is used to resolve conflicts for the data services.
3. Reduction in query read ahead, which compares the priority of the currently active request with the priority of any active disk I/O. This is used to resolve conflicts for I/O services.
4. Query deferral, which compares the priority of the current active request with the priority of any active disk I/O. This is used to resolve conflicts for I/O services when parallel query workloads are active.

## **Contention for CPU Resources**

Various techniques are used to ensure high priority workloads are not impacted by CPU intensive low priority workloads such as SQL query activity. The message queue for DP2 services is priority ordered, and the DP2 process will wait on the ready-list at the request priority until it reaches the head of the ready-list. Once DP2 is dispatched, it will process the request at the normal DP2 priority (220).

If the current request is a lower priority SQL query, DP2 will periodically yield the CPU to processes with a priority higher than the currently active request. The frequency is limited to avoid the extra overhead of context switching.

## **Contention for DP2 Resources**

In addition to the check for CPU contention, DP2 will also check the message queue to determine if there is contention for data access services. If there is a new request for service with a higher priority than the currently active request and it is not handled by another process in the group, then the current request will be preempted.

## **Contention for I/O Resources**

SQL query workloads can be highly optimized with predicate evaluation performed directly by the data access manager. This is most notable with a query that consumes all data in the table. Such a query can be throttled in the face of higher priority work by reducing the rate of read-ahead.

The amount of reduction in read-ahead is limited. If the read-ahead is reduced to zero (single block I/O), the impact to the I/O system can be as bad as too much read-ahead in most cases.

## **Query Deferral**

Simple contention checks are not sufficient to prevent impact to high priority transaction response time if the amount of resources typically consumed by high priority workloads is below capacity. Any available resources that allow some service to lower priority workloads can result in some transaction response time delays due to the CPU and I/O context switching.

To overcome this problem, lower priority SQL query workloads may be periodically deferred by DP2. The duration of deferral is directly related to the priority of the query. The lower the priority, the longer the deferral period, and the higher the priority, the shorter the deferral period.

The deferral can be viewed as a time slice. SQL query workloads are typically processed by the data access manager at one-second intervals, where DP2 processes work during that period, with a re-drive mechanism between the SQL file system and DP2 every second. The interval may be smaller for contention, buffer overruns, and other conditions, such as lock conflicts.

If the priority of the query is above 150, a DP2 process will spend the entire 1-second interval working on the query before invoking the re-drive. If the priority is 150 or less, DP2 will delay work on the query for some fraction of the 1-second interval. The delay period is inversely proportional to the request priority.

This means that the relative priority of the SQL query acts as a throttle in the face of higher priority transaction workloads. The lower the priority of the SQL query, the less impact to transaction response time, and the higher the priority of the SQL query, the more impact to transaction response time.

At the same time, the SQL query run in the face of higher priority transaction workloads will run faster and consume more resources when run at higher priority, and will run slower and consume less resources when run at lower priority.

## Priority Guidelines

Prior to determining guidelines for setting priorities of SQL queries, it is necessary to measure and establish response-time requirements for high-priority transactions. It is then possible to fine-tune the priority of SQL query workloads to determine the best priority.

If response time requirements cannot be measured or established, then the only recourse is to run the SQL query at a very low priority to ensure there is no impact to higher priority workloads. While the SQL query may run significantly slower in the face of higher priority workloads, it may be possible to disable parallel execution for the SQL query (CONTROL QUERY PARALLEL EXECUTION OFF) and run the SQL query at a higher priority in some cases.

There are new MEASURE counters available to provide some indication of the amount of query deferral for each DP2 volume. The DEFREQ-QTIME counter is the amount of time a request is subject to deferral, and the DEFERRED-QTIME counter is the amount of time a request is deferred.

The ratio between these two counters should increase as the priority of the SQL query is increased. Because SQL query workloads can vary significantly in resource consumption, and because high priority workloads and transaction response time requirements can vary, it is difficult to provide general guidelines.

The performance data published with each major release includes data for mixed workload tests. These include SQL query workloads run at various priority schemes concurrent with higher priority ORDER-ENTRY benchmarks. While these numbers can be used as a guide, actual results will typically be different due to the difference with ad-hoc query workloads and the variability of high priority workloads.

In general, when there are multiple partitions processed by a parallel scan within the same CPU, the more likely the requirement that the priority of the query should be reduced to avoid impact to high priority transaction response time.

Starting with a lower priority (5-10) and measuring results may be preferable to starting higher (150) and reducing, especially during live system application activity.