Scaling Out Query Performance with SAP® Sybase® IQ PlexQ™
A Shared Everything Architecture with Data Affinity for Massively Parallel Processing

Principal Author: Courtney Claussen, Senior Product Manager, SAP Sybase IQ Product Management, SAP
Contributing Authors: Kurt Deschler, Development Manager, Sybase IQ Query Engine Product and Technology Operations, and Peter F. Thawley, Senior Director/Architect, Office of the CTO

Version 2.0, July 2013
TABLE OF CONTENTS

INTRODUCTION ................................................................................................................................................................. 3
  Sybase IQ PlexQ — A Multiplex Foundation .................................................................................................................. 3
  Balancing the Benefits of Parallelism with the Risk of Saturation ............................................................................. 4
  The Importance of SAN Performance in DQP ............................................................................................................ 5

UNDERSTANDING PARALLELISM IN SYBASE IQ ........................................................................................................ 5
  Query Processing and Data Flow Model ......................................................................................................................... 5
  Dynamic Parallelism ........................................................................................................................................................ 6
  Intra-operator Parallelism Enhancements in Sybase IQ 16 .......................................................................................... 7

UNDERSTANDING DISTRIBUTED QUERY PROCESSING .................................................................................................. 7
  What is DQP? .................................................................................................................................................................... 7
  How Does DQP Work? ................................................................................................................................................... 7
  Hash Partitioned Tables and Data Affinity in SAP Sybase IQ 16 .................................................................................. 9
  Distributed Query Processing within Logical Servers .................................................................................................. 10
  Multiplex Inter Node Communication ........................................................................................................................ 10
  Pre-requisites for DQP .................................................................................................................................................... 11
  What Kinds of Queries Can Be Distributed Across a PlexQ Grid? .............................................................................. 11
  How Do You Know if a Query Was Distributed? ........................................................................................................... 12
  How Are Errors Handled? ............................................................................................................................................... 15

SCALABILITY OF DQP .............................................................................................................................................. 16
  Queries Highly Likely to Benefit from DQP .................................................................................................................... 16
  Queries Generally Unlikely to Benefit from DQP ........................................................................................................ 16
  What You Can Do to Influence DQP Scalability .......................................................................................................... 17
  Sizing Shared Temporary Storage ................................................................................................................................ 17

DQP SINGLE QUERY WORKLOAD TEST RESULTS ................................................................................................... 18
  Query A .............................................................................................................................................................................. 19
  Query B ........................................................................................................................................................................... 20
  Query C ........................................................................................................................................................................... 21

SUMMARY AND CONCLUSIONS .................................................................................................................................. 21
INTRODUCTION

SAP Sybase IQ has a Massively Parallel Processing (MPP) architecture called PlexQ™ that accelerates highly complex queries by distributing work to many computers in a grid. Unlike shared nothing MPP architectures, PlexQ utilizes a shared everything approach that dynamically manages and balances query workloads across all the compute nodes in a Multiplex grid. PlexQ’s automatic workload re-balancer aggressively works to avoid contention among users for system resources, thereby providing predictable high performance and resource efficiency for a broad spectrum of concurrent workloads. Shared everything architectures have the benefit of a shared store that all nodes can access freely. Without storage hardware that supports high I/O bandwidth, however, a shared store can be susceptible to I/O bottlenecks as multiple machines read from and write to the same disks. Shared nothing architectures, when data is balanced properly among all nodes, can achieve better performance than shared everything architectures, because there is no disk contention. To achieve similar performance while still maintaining the flexibility of a shared store, SAP Sybase IQ 16 introduced a feature called data affinity. Data affinity allows individual data sets to “prefer” the cache of a particular server as queries execute. The result is better performance because data tends to reside for longer periods of time in memory.

At the heart of PlexQ is an exciting and broadly applicable capability commonly known within the database community as “Distributed Query Processing”, or DQP for short. DQP can improve the performance of a query by breaking it up into pieces and distributing those pieces for concurrent execution across multiple SAP Sybase IQ servers in a grid. This approach builds on the “scale up” parallel processing model initially delivered in SAP Sybase IQ 15.0 by adding a “scale out” parallel processing model to leverage more independent compute resources for faster answers to the increasingly complex and time-critical business questions that must be met by IT departments under strict Service Level Agreement (SLAs). With SAP Sybase IQ 16 data affinity, distributed query processing is intelligently managed by assigning query work on a particular data set, to the machine that already has that data set in cache. Caches stay hot, I/O is reduced, and SAP Sybase IQ’s shared everything architecture operates more like a massively parallel processing shared nothing architecture.

As technologists, it is easy for most of us to get caught up in the excitement of such a promising enhancement. As you will see when you begin to explore this feature yourself, DQP can dramatically speed up many queries, not to mention doing so cost effectively by leveraging all of the existing compute resources in a cluster rather than forcing yet another hardware upgrade to get the speeds and feeds of the latest systems. However, it is important to understand that it is not as simple as some vendors’ marketing departments might lead you to believe — there is a reason they call it “computer science!”

This paper is meant to give you a holistic perspective as we introduce the technical concepts behind DQP in general and SAP Sybase IQ 16.0’s DQP implementation in particular. The remainder of this technical white paper describes parallelism and DQP in general, explores the systemic considerations that are key to attaining the speed-up and scale-out benefits desired, discusses the types of queries it most benefits, and finally quantifies some of the performance improvements observed in the lab so far.

SAP Sybase IQ PlexQ — A Multiplex Foundation

In the mid 1990’s, Sybase pioneered the concept of column-oriented databases with its Sybase IQ product. At the time, column-oriented database technology was primarily debated within the academic community but as data volume requirements began to explode in the early 2000’s, the true value of column stores became more evident. Originally architected to support heavy ad hoc queries and large numbers of concurrent users, Sybase IQ took a hybrid approach when it came to clustered configurations, which it called Multiplex.

Some vendors, especially the up-start “parallel databases” took the academically pure “shared nothing MPP (Massively Parallel Processing)” approach where data is physically partitioned across a number of independent servers, each with their own memory and storage subsystems. While this model can provide good performance, the administrative and maintenance challenges are non-trivial. By far the greatest operational challenge for this purist approach isn’t observable during an initial trial or proof-of-concept. It only surfaces after months when data distributions across the different independent nodes naturally starts to skew such that the administrative overhead of keeping relatively balanced amounts of data on each node is nearly an impossible feat requiring extensive monitoring and data movement during the ever-shrinking maintenance windows that customers face. In addition, as more concurrent users come online in production environments, the shared nothing architecture tends to slow down considerably due to system saturation by multiple users and bottlenecks arising out of a single path through one ‘master’ node for all concurrent queries.
For these reasons, SAP opted for a “shared disk cluster” model for both SAP Sybase IQ and its OLTP-focused sibling, SAP Sybase Adaptive Server® Enterprise (ASE), and now extended SAP Sybase IQ with this “shared everything MPP” model. This approach makes perfect sense for analytic and data warehousing workloads because it allows data center operations to scale-out data storage volume and concurrent user request volume independently thereby balancing great performance with operational simplicity. This approach also appeals to most companies because it leverages their existing capital and staff skill set investments in highly resilient SANs (Storage Area Networks).

**Balancing the Benefits of Parallelism with the Risk of Saturation**

Like the laws of physics, there are certain characteristics of both the computer hardware and software layers of a systems’ stack that are relatively immutable, at least with respect to the current products and technologies you own. In the storage layers, disks (spindles, flash, etc.) each have a maximum number of operations per second (IO’s/Second, a.k.a. “IOPS”) they can service simply due to the mechanics (or physics) of the technology or a hardware vendor’s specific implementation. Similarly, Host Bus Adapters (HBAs) and other disk controllers all have a maximum bandwidth (throughput measured in MBs/Second) of data they can transfer. In the networking layer, different networks (1GbE, 10GbE, Infiniband, etc) and Network Interface Cards (NICs) have effectively the same types of limits.

Different processor architectures (e.g., Intel Nehalem, IBM POWER7, Sun UltraSPARC and SPARC64, etc.), not to mention processor models, all have differing numbers, sizes, and sharing capabilities for the on-chip resources such as caches (L1, L2, and L3) and execution units. In addition, they each have different memory interconnect architectures, often with huge differences in both bandwidth and latency. With the processor trends implementing lighter weight hardware threads (a.k.a. strands) within the processor itself, it is not surprising to see different processor architectures and models offer both differing amounts of parallelism as well as different “quality” of parallelism.

Not to pick exclusively on the hardware, it should not surprise you that the software layers — from the DBMS, through the operating system, and into the device drivers — also impact performance, often significantly. For example, some database management systems’ (DBMS), especially traditional row stores, place constraints on parallelism by mapping its usage to tables’ partitioning characteristics. DBMS’s and operating systems often have limits on I/O, both in terms of size but also in concurrent numbers, which could impact performance. Most importantly, at least for this discussion, is that SQL queries are not all created equal — some lend themselves to parallelism quite naturally, others do not! Fundamentally, this is the key issue that you must understand in order to set the right expectations with your business users. The following two examples should help give credence to this fact.

First, consider a query which scans a billion rows applying simple filtering types of predicates (i.e., where clauses). If the selection criteria is such that the DBMS must return tens or hundreds of thousands of rows to the application, does it makes sense to parallelize the query? The answer is no. Why you ask? Well, all of those rows have to be sent back to requesting application though a single TCP/IP connection, which implies a single DBMS instance on one of the PlexQ nodes. The time to funnel all the data back to a single node and send the result set back to the application may dominate the overall execution time of the query, thus rendering little or no value from distributing the query across the PlexQ grid.

A less obvious example is when a query is “i/o bound” — that is, the time waiting for physical I/O is significantly higher than the time needed to perform calculations. If threads execute for only a brief amount of time, say 10’s – 100’s of microseconds (1 µs = 1 millionth of a second) before it must wait several milliseconds (1 ms = 1 thousandth of a second) for an I/O to return, there is little to be gained from the overhead of distributing the workload across the cluster because the threads use so little CPU time as a percentage of the overall query execution time.

One of the challenges to enabling parallelism and DQP on a system is the risk that it could saturate components of the system. This most often comes into play during times when the system is servicing large numbers of concurrent queries from multiple users, each of which then spawns multiple threads to complete requests as quickly as possible. With more traditional databases, parallelism is often quite static — numbers of threads are fixed and unchanging because they were assigned as part of the optimization phase. While customers should certainly monitor their systems for saturation, SAP Sybase IQ’s PlexQ platform helps to minimize this risk by implementing an advanced and dynamic model of parallelism as a run-time optimization which can be scaled up or back depending on the current resource availability of the system and workload demands. This innovation operates even within the execution of a single query to ensure the best possible performance and resource utilization irrespective of when a request is made (i.e., during a highly concurrent online time period or at low concurrent times such as nightly batch reporting).
The Importance of SAN Performance in DQP

Since SAP Sybase IQ’s data is stored centrally on network-attached storage, most often an enterprise-class SAN, this becomes a critical resource that can make the difference between meeting and missing an application’s service-level agreement (SLA) with its’ business users. From a performance perspective, general parallel and DQP configurations tend to allocate more simultaneous work from a larger number of concurrent threads performing work in support of applications’ queries. It is this volume of concurrent work that often can stress the systems’ components. Often times, components in the SAN are the first to show distress as they reach various saturation points.

The good news is that SANs are very extensible, in terms of both storage capacity as well as raw performance. In addition, most storage teams already have the tools and skills to identify and resolve storage performance problems, often before they become perceptible to the business users of these applications. As companies begin to utilize and deploy increased parallelism and DQP configurations in support of their business objectives, it is highly recommended that they closely involve their storage teams to ensure the storage subsystem has sufficient performance, both in terms of IOPS and bandwidth (MB/Sec) to support the requirements of the business.

UNDERSTANDING PARALLELISM IN SAP SYBASE IQ

Query Processing and Data Flow Model

Most traditional databases create a base table of data, stored as sequential rows of contiguous columns. In Sybase IQ, the columns of a table are stored separately from each other, and a row is only a virtual entity, until it is constructed dynamically during the course of running a query.

Like any other database, SAP Sybase IQ accepts a query from a front-end tool, parses it, and then passes the parsed query to the SAP Sybase IQ optimizer. While SAP Sybase IQ is optimizing a query, it builds a “tree” of objects (joins, group by clauses, sub-queries, etc.) Tables are “leaves” at the bottom of the tree, and rows of data flow up the tree from the leaves to a “root” query node at the top, where the data is passed from SAP Sybase IQ to the requesting user.

The data flow tree begins execution at the root query node. It starts by requesting a first row from the query node below. This child query node “wakes up” and begins asking for rows from the next query node below it. This continues down the tree until the execution reaches the leaf query nodes, which read the actual data from the tables. The figure to the right depicts this common approach.

A leaf query node performs two functions in a SAP Sybase IQ query. First, it processes the local table predicates — the parts of the WHERE clause that access only one table. These local predicates are processed vertically, meaning that individual columns are evaluated individually using the SAP Sybase IQ indexes. The second function is to project the set of rows that satisfy all the conditions of the local predicates up to the leaf’s parent node. The data is now processed horizontally, as rows (tuples).
SAP Sybase IQ supports two types of parallelism as it processes a query:

- **Inter-operator parallelism**: multiple query nodes in the query tree execute in parallel
- **Intra-operator parallelism**: multiple threads execute in parallel within a single query node

**Inter-operator parallelism** is accomplished using two different parallelism models: pipeline parallelism, and bushy parallelism. With pipelining, a parent node can begin consuming rows as soon as a child node produces its first row. With bushy parallelism, two query nodes are independent of each other and can execute in parallel without waiting for data from each other.

**Intra-operator parallelism** is accomplished by partitioning the operator’s input rows into subsets, and assigning the data subsets to different threads. SAP Sybase IQ makes heavy use of both inter- and intra-operator parallelism to optimize the performance of queries.

### Dynamic Parallelism

Parallelism allows maximum utilization of resources to improve the performance of a query. However, it is often undesirable for one “big” query to starve out other queries running at the same time. The SAP Sybase IQ query engine adapts to changes in server activity by increasing or decreasing parallelism dynamically. For example, a resource intensive query running alone might use many or all of the CPUs, now potentially on all the servers in the PlexQ grid. Then, as other users start queries, even while the first query is still running, SAP Sybase IQ will gracefully scale back CPU resources (threads) and their associated memory, dynamically allocating them to these new queries! As these other queries complete, their resources can be reallocated back to queries that are currently running so they leverage more compute resources to complete faster.

Figure 2 below illustrates this by showing how as different queries are started, the CPU resources used by each are reduced to ensure the total system does not over commit the total resources in use and become saturated to the point when the entire system starts the “thrash”. Equally valuable though, when CPU availability increases because queries complete, these resources are made available almost immediately to leverage the idle capacity and allow the running queries to complete as quickly as possible.

![Figure 2 – Balancing Parallelism and Resource Availability](image)
Intra-operator Parallelism Enhancements in SAP Sybase IQ

Over the course of a few releases, SAP Sybase IQ has significantly enhanced intra-operator parallelism. Most query operations can now be performed in parallel using many threads:

- Table join operations
- Group by operations
- Sorting (Order By and Merge Joins)
- Predicate execution in tables: e.g., “WHERE last_name like “%son%”, range predicates, IN conditions, “Top N” operations, and many others

Prior to SAP Sybase IQ 15.3, inter-node and intra-node parallelism within a single query could only use the CPU resources on a single server. During that time, SAP Sybase IQ Multiplex configurations were a very effective way to scale up support for more and more concurrent users or queries but it did nothing to reduce query execution times by leveraging all the compute bandwidth across the grid. SAP Sybase IQ PlexQ lifted that restriction, allowing a query to use the CPU resources on potentially all the machines in the grid.

Let’s now explore the details of how DQP works in a SAP Sybase IQ PlexQ shared everything MPP architecture.

UNDERSTANDING DISTRIBUTED QUERY PROCESSING

What is DQP?

Distributed Query Processing (DQP) spreads query processing across multiple servers in a SAP Sybase IQ Multiplex grid. A SAP Sybase IQ Multiplex is a group of servers, each running SAP Sybase IQ. The servers in a grid connect to a central store, such as a shared disk array, for permanent shared data. SAP Sybase IQ PlexQ is the hybrid cluster architecture of a Multiplex that involves shared storage for permanent data, and independent node storage for catalog metadata, private temporary data, and transaction logs.

When the SAP Sybase IQ query optimizer determines that a query might require more CPU resources than are available on a single node, it will attempt to break the query into parallel “fragments” that can be executed concurrently on other servers in the grid. DQP is the process of dividing the query into multiple, independent pieces of work, distributing that work to other nodes in the grid, and collecting and organizing the intermediate results to generate the final result set for the query.

It is important to emphasize that if a query does not fully utilize the CPU resources on a single machine, then it will usually not be advantageous to distribute it. For example, if the optimizer is going to parallelize a query 7 ways (keep 7 threads at a time busy) on an 8 core box, it will probably not distribute it. Distribution requires network and storage overhead to assign work, and store and transmit intermediate results. The objective in a DBMS is to execute queries as quickly as possible. A simple query will run fastest on a single machine. However, large and complex queries that can exceed the CPU capacity on a machine may be better served by incurring the overhead of distribution.

If performance is improved, then distribution is a win.

How Does DQP Work?

DQP is available to any customer who has deployed an SAP Sybase IQ Multiplex grid. When you install SAP Sybase IQ, DQP is turned on by default, and all servers in the grid may be utilized for distributed processing.
DQP introduces the concept of “Leader” and “Worker” nodes. The Leader node is the node where a query originates. A Worker node can be any node in the grid that is capable of accepting distributed query processing work. All grid node types (readers, writers, or coordinator) may serve as Leader or Worker nodes:

![Figure 3 – A Distributed Query in Action](image)

In the picture above, execution of query 1 and query 2 is distributed across subsets of nodes in the PlexQ grid. The two queries are serviced by different leader nodes and sets of worker nodes. This is one possible operational scenario. You can configure the set of nodes that participate in a distributed query very flexibly (see “Logical Servers” below).

SAP Sybase IQ also requires a new shared DBSpace, called Shared Temporary Store, to support DQP. This DBSpace is named IQ_SHARED_TEMP, and must reside on shared disk storage accessible and writeable by all nodes in the grid. These are the same requirements that exist for IQ_SYSTEM_MAIN and user defined DBSpaces for user data. The purpose of IQ_SHARED_TEMP is to allow transmission of intermediate data in both directions for servers involved in a distributed query. IQ_SHARED_TEMP and the local temporary store, IQ_SYSTEM_TEMP, both use the temporary cache for in-memory buffering of data.

To offset the requirement for additional IQ_SHARED_TEMP shared storage, SAP Sybase IQ 16 introduced a new feature called “DQP over the network”. This feature allows intermediate results generated during distributed query processing to be sent over the network, instead of using IQ_SHARED_TEMP. If you have a fast network, you might want to choose this option, instead of configuring shared disk to transmit results. Make sure you have enough cache to maintain the intermediate results in memory, otherwise they will spill to disk, and will have to be re-read from local IQ_SYSTEM_TEMP before being shipped over the network.

When a client submits a query to the SAP Sybase IQ server, the query optimizer uses cost analysis to choose whether to parallelize and/or distribute execution of the query. A parallelizable query is broken into query fragments — predicates and data flow sub-trees. A query fragment is considered eligible for distribution only if the SAP Sybase IQ engine supports parallel and distributed execution of all of the query operators contained in the fragment.
When a query is distributed, the leader node assigns query fragments to workers, and collects intermediate results from the worker servers. Workers do not make decisions about query distribution. They simply execute the work assigned to them and return results.

If the query optimizer makes a determination that a distributed query will not scale appropriately, or might even degrade in performance, then the query will not be distributed and will be executed on a single node in the grid. Queries are classified as follows:

1. Not distributed: no fragments are executed on other nodes of the PlexQ grid. The query is run completely on the leader node.
2. Partially distributed: one or more fragments are executed on other nodes of the PlexQ grid as well as the “Leader Node”.
3. Fully distributed: all fragments are executed on multiple nodes of the PlexQ grid.

Hash Partitioned Tables and Data Affinity in SAP Sybase IQ 16

Hash partitioned tables were introduced in SAP Sybase IQ 16 to pre-partition data at load time. This is beneficial for the query engine, which wants to parallelize and distribute work for better performance. Work can be parallelized if at least two processing threads can work on disjoint data sets at the same time, and then aggregate the results later. In a Multiplex, not only can the work be parallelized, but distributed across a set of machines. When a table is “hash partitioned”, its data is divided up into 32K “buckets” that are accessed with a hash key. The hash key is defined by a column or set of columns in the table, which define a unique key to each row. The query engine can operate on these smaller buckets of data, often using more efficient hash join and group by algorithms. Sorts can also be distributed by sorting the smaller sets independently, and then merging the sorted results later. This is often faster than a large, monolithic sort. Some new partitioned join and group by algorithms have also been introduced in SAP Sybase IQ 16 to work specifically on partitioned data. The result is that hash partitioned tables may not need additional, bitmapped indexes such as DATE/DTTM and HNG to improve performance.

In order to determine which tables to hash partition, the user should review query plans that exhibit slow performance, and choose valid hash partition keys on joined tables that might speed processing. Hash partitioning incurs overhead at load time, so you may not want to hash partition every table in your database. Note that the new partitioned join algorithms will be selected by the query optimizer only if all tables involved in the join are hash partitioned, and the respective hash keys involved in the join operation have the same data types. Another benefit of the new partition-based algorithms, is that because they are operating on smaller data sets, they tend to use less temp cache.

Data affinity is software-based work allocation that improves cache efficiency in a Multiplex during distributed query processing. Data affinity associates data partitions with certain nodes in a cluster. Partitions may be hash partitions of a hash partitioned table, or row ranges in a non-hash partitioned table. Once the partition association with a server node has been made, the query engine assigns work based on that partitioning. Once a steady state is achieved, the buffer caches tend to have an even distribution of the data. However, if a node completes its work early, it may “steal” work from another node. As queries run, statistics around stealing and work assignments are gathered and sent to the coordinator. The coordinator manages an “affinity map” which maps data partitions to nodes. If a node is routinely not completing its work at the same pace as other nodes, the coordinator will slowly adjust the affinity map, to favor faster nodes with partitioned work. Faster nodes over time will be assigned more partitions.

In order to keep nodes that are stealing work from invalidating their own affinitized pages, there is a server option called cache_affinity_percent, that may be set from 0 – 100 (default is 70). This value sets the maximum percentage of main cache to use for affinity. Non-affinity, or stolen data, can occupy the rest of the cache. The following picture shows the benefit of affinity for the TPC-H workload. The lines show the amount of I/O per query in GB against the shared store. Prior to SAP Sybase IQ 16, much of the data could be duplicated across the nodes, because work was distributed randomly. With data affinity, work is distributed to the nodes that already have the data in cache, and the caches have much less duplication of data across nodes.
You will see indicators in the SAP Sybase IQ query plan about percentages of work assignments that were affinitized and non-affinitized. You will also see “orphaned” work percentages in the query plan. Orphaned work indicates work that should have been affinitized to a node, but the leader was quicker than the assigned node, and accomplished the work itself.

**Distributed Query Processing within Logical Servers**

You may not always want to use all the servers in a grid for distributed query processing, and may want to provision a subset of these resources by application or user. For this purpose, SAP Sybase IQ has the concept of a logical server. A logical server allows one or more servers in a grid to be grouped together and represented as a logical entity. You can dynamically add or drop logical server member servers to accommodate the changing resource needs of applications. Users are granted access to logical servers via the login policy associated with the user.

There are some built-in logical servers. In particular, the built-in OPEN logical server includes all servers that are not members of any user-defined logical server. If you do not create any logical servers, all nodes in the grid may participate in DQP, because they are part of the OPEN server.

A user’s login policy specifies which logical servers the user is allowed to connect to, and run distributed queries on. When a user connects to a Multiplex, he is effectively connecting to a logical server. Logical servers have associated logical server policies. One attribute of a logical server policy is “Login_redirection”. This means that when a user connects a particular SAP Sybase IQ server, the user under certain conditions may be redirected to a different physical server than the one he explicitly connected to. One condition is that if the user is connecting to a physical server that is down, or not part of the user’s allowed logical servers, the user will be redirected to a different physical server that is a member of the user’s default logical server, or another logical server he is allowed to connect to. Another condition is that if a server is overloaded – more clients are waiting to connect to the server than are allowed by the –rgovern option – then the user will be redirected to a less busy physical server in the same logical server. Finally, in a connection string, the user may specify that he wants to connect to a reader node or a writer node. If the physical server he is connecting to is not of the type he specifies, SAP Sybase IQ will redirect the user to a physical node with the correct role.

**Multiplex Inter Node Communication**

In order to support streamlined communication among nodes participating in the distribution of a query, SAP Sybase IQ introduced the Multiplex Inter Node Communication (MIPC) framework. The MIPC mesh is a peer-to-peer inter-node communication infrastructure that supplements the Inter Node Communication (INC) protocol added in IQ 15.0. INC is
used for two-way heartbeat monitoring, version data synchronization, and other types of message and data propagation required in a PlexQ grid. INC allows nodes to talk to each other only via the coordinator, and was adequate for the more limited communication requirements of single node queries. MIPC allows PlexQ grid nodes to talk directly with each other, and supports the more robust communication requirements of DQP.

There are both public and private configuration options for the MIPC. The private option allows you to specify host-port pairs (TCP/IP protocol only at this time) that PlexQ grid servers will use exclusively for DQP related communications. If no private interconnection configuration is provided, MIPC uses the host-port pairs specified for other types of communication: external user connections and INC connections.

A private MIPC network has been found during internal testing to provide significant performance benefits over a shared MIPC network — in one particular instance, a distributed query running on 2 nodes over a private MIPC network executed almost as quickly as a 3 node configuration using a shared MIPC network.

Pre-requisites for DQP

You do not need to set any configuration options to activate distributed query processing. Unless you disable DQP by turning off the dqp_enabled logical server policy option, DQP occurs automatically for qualifying queries when:

• The server is part of a PlexQ grid.
• There is a logical server with login permissions, and at least one node available. By default, there is a built-in logical server called the OPEN logical server, so this requirement is satisfied out of the box.
• The shared temporary DBSpace has writeable files available. Initially there are no DBFiles in the shared temporary DBSpace, and the PlexQ grid administrator must add at least one raw device DBFile to it in order to activate distributed query processing. Note that if you have specified the logical server policy option to perform DQP over the network, you won’t need a shared temporary DBSpace.

What Kinds of Queries Can Be Distributed Across a PlexQ Grid

In order for a query operator to be distributed, it must be able to be executed in parallel. When an operator is executed in parallel, multiple threads can be applied to execute the processing in parallel. In SAP Sybase IQ, most query operators can be parallelized but not all are distributed.

The following table shows which query operators are distributed:

<table>
<thead>
<tr>
<th>CLASS</th>
<th>OPERATOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOIN</td>
<td>Nested Loop / Nested Loop Pushdown Hash / Hash Pushdown</td>
</tr>
<tr>
<td></td>
<td>Sort Merge / Sort Merge PushDown</td>
</tr>
<tr>
<td></td>
<td>Asymmetric Sort Merge / Sort Merge PushDown</td>
</tr>
<tr>
<td></td>
<td>Asymmetric Hash / Hash PushDown</td>
</tr>
<tr>
<td></td>
<td>Partitioned Join algorithms</td>
</tr>
<tr>
<td>GROUP BY</td>
<td>GROUP BY SINGLE</td>
</tr>
<tr>
<td></td>
<td>GROUP BY (HASH)</td>
</tr>
<tr>
<td></td>
<td>GROUP BY (SORT)</td>
</tr>
<tr>
<td></td>
<td>Partitioned GROUP BY algorithms</td>
</tr>
<tr>
<td>DISTINCT</td>
<td>DISTINCT (HASH)</td>
</tr>
<tr>
<td></td>
<td>DISTINCT (SORT)</td>
</tr>
<tr>
<td>SORT</td>
<td>ORDER BY</td>
</tr>
<tr>
<td></td>
<td>ORDER BY (N)</td>
</tr>
<tr>
<td></td>
<td>SORTED IN</td>
</tr>
<tr>
<td>SUBQUERY</td>
<td>Uncorrelated</td>
</tr>
<tr>
<td>PREDICATES</td>
<td>Condition Execution (using FP / LF / HG indexes)</td>
</tr>
<tr>
<td>OLAP</td>
<td>OLAP RANK and WINDOW with PARTITION</td>
</tr>
<tr>
<td>SELECT component of INSERT operations</td>
<td>INSERT...SELECT INSERT...LOCATION</td>
</tr>
</tbody>
</table>
Query fragments that have the following behavior are never distributed:

1. Write to the database (including DDL, INSERT, LOAD, UPDATE and DELETE)
2. Reference temporary tables
3. Reference tables that reside in the SYSTEM DBSpace
4. Reference proxy tables
5. Utilize non-deterministic functions, such as NEWID

Note that a LOAD operation can still be “distributed” by loading individual tables in parallel using multiple writer nodes in the grid.

How Do You Know if a Query was Distributed?

The SAP Sybase IQ query plan gives you visibility into whether or not a query was distributed. The query plan provides detailed information that indicates which servers participated in the query processing, measures how the work was distributed, and displays timing information.

DQP begins when a client connects to a physical server and initiates a query. This server is the leader node for the query. The leader node invokes the query optimizer to build the execution plan for the query. The query optimizer builds a query tree, and divides the query into fragments. A fragment is either:

1. A leaf condition (a predicate)
2. A data flow sub-tree with a particular partitioning: range of rows or keys

Fragments are portions of the query tree that can be executed independently. When two fragments may execute in either order, they may execute concurrently. If one fragment depends on intermediate results from another fragment, then the two must execute in the proper order. If all the query operators in a fragment are parallelizable and distributable, then the fragment is eligible for distribution across all the worker nodes. A fragment that cannot be distributed will execute completely on the leader node. The optimizer divides each query operator in a fragment into a set of “work units”. A work unit is a subset of data for a processing thread to work on.

Here is an example of a query plan broken into query fragments. You will not actually see the dotted lines shown below in the real query plans. This just gives you a feel for how a query might be fragmented by the optimizer. In this example, fragments 1, 2, and 3 will execute concurrently:

![Figure 5 – Sample Query Plan Illustrating Distributed Processing Fragments](image-url)
When you turn on the database options to create query plan files, the query plan for the whole query will be created on the leader node. When a query fragment is a data flow sub-tree, and it is distributed, each worker that participates in executing the fragment will generate a local query plan for that fragment. (Note that you need to turn on the query plan database options only on the leader node, not the worker nodes, for query fragment plans to be created on the workers.) The query operator at the top of a fragment manages the assignment of the fragment’s work units to threads across all the workers.

Allocation of threads to work units is a highly dynamic process that allows threads to be added and removed from a query as it executes. Threads are scaled up and down based on machine load and resource availability. Availability of temp buffers and CPU time are the dominant factors in decisions to add or remove threads. In SAP Sybase IQ DQP, physical servers can be added to a logical server dynamically, and after some initialization, can begin performing DQP work as soon as a new query fragment is assigned for distribution.

![Sample Query Plan Illustrating Distributed Processing Fragments](image)

If a part of a query is distributed, you will see a triple black line between nodes that were distributed. When you hover a mouse cursor over the row count next to the parallel lines in the display, it will show the number of remote rows (how many were distributed). The width of the rightmost bar is sized depending on the number of remote rows.

Below the query tree, is the timing diagram. At the top, for each node in the query tree, you will see timings for each phase of its execution. This now includes timings across all the servers in the grid. The CPU utilization portion of the timing diagram will also show aggregated times across all servers.
Below the node phase timings is the threads display. This shows which threads on which servers are performing work at a particular time. Thread assignments are shown as a stacked bar graph:

![Graph showing threading and distribution across the PlexQ Grid](image)

Figure 7 – Query Plan Section Showing Threading and Distribution Across the PlexQ Grid

If you mouse over a thread block, you will see various statistics, such as:

- #53: The number of the node at the root of the query fragment that is executing
- S:2: Server ID (2) that owns the threads doing the processing
- T: 0–3: Range of threads doing the processing
- A:2: Average number of threads (2) doing the processing during that slice of time
- N:3: Number of samples taken (3) to calculate thread statistics during that slice of time
- 23:25:13... - 23:25:14: Start and end times of the time slice

If a query fragment is executing on multiple servers at the same time, you will see thread blocks for the same root node of the fragment stacked above each other.
Below the timing diagram are node specific details:

**Fragment 1**

- **#15 Parallel Combiner**
  - Parent Node: 0002
  - Child Node: 0006
  - Generated Result Rows: 40
  - Estimated Result Rows: 1
  - Work Units - kwd_nc16165: 25 (2, 6, 4, 4, 3, 2, 3, 1)
  - Work Units - m20242_kwd: 24 (4, 3, 3, 3, 3, 3, 1)
  - Max. Possible Parallel Arms: 16
  - Max. Active Parallel Threads: 16
  - Parallel Sink Work Units: 40
  - First Worker Work Unit: 2

**Fragment 2**

- **#48 Order By (Parallel inputs and outputs)**
  - Parent Node: 0001
  - Child Node: 0001
  - Generated Result Rows: 9076814
  - Estimated Result Rows: 983538
  - Work Units - kwd_nc16165: 15 (0, 2, 3, 3, 2, 2, 1, 0)
  - Work Units - m20242_kwd: 17 (3, 3, 3, 3, 3, 3, 2, 2)
  - Max. Possible Parallel Arms: 16
  - Max. Active Parallel Threads: 14
  - Parallel Sink Work Units: 92
  - First Worker Work Unit: 2

**Fragment 3**

- **#53 Order By (Parallel inputs and outputs)**
  - Parent Node: 0003
  - Child Node: 0002
  - Generated Result Rows: 2221520
  - Estimated Result Rows: 2221520
  - Work Units - kwd_nc16165: 10 (0, 3, 3, 2, 2, 1)
  - Work Units - m20242_kwd: 8 (1, 2, 2, 2)
  - Max. Possible Parallel Arms: 16
  - Max. Active Parallel Threads: 8
  - Parallel Sink Work Units: 16
  - First Worker Work Unit: 1

*Figure 8 – Query Plan Detailing Node-specific Information*

For a particular node, you will see how work was distributed to servers and threads. In “Fragment 1” above, the value of work units for server “kwd_nc16165” is “25 (2, 6, 4, 4, 3, 2, 3, 1)”. This means that 25 work units were assigned to this server, and 2, 6, 4, 4, 3, 2, 3, and 1 of those work units respectively were assigned to 8 different threads. You can also see how much private and shared temp space was used to execute the fragment.

“Fragment 2” shows the number of the first work unit assigned to a worker. A number greater than 1 means that the leader executed some work first, before a worker was able to begin processing. This is probably due to a delay getting the worker what it needs to begin performing work.

“Fragment 3” shows “Parallel Sink Work Units” which are the total number of work units for the entire fragment.

**How are Errors Handled?**

DQP is tolerant to worker/network failures and slow workers. If a worker node fails to complete a work unit due to an error or a timeout violation, the work unit is retried on the leader node. If this occurs, the worker node will be given no more work units for the duration of the fragment execution.

Although a worker might fail while executing work for one query fragment, it may still be assigned work units for a different query fragment later in the process.
SCALABILITY OF DQP

A query is likely to benefit from DQP only if it is fully parallel and CPU bound on a single node. In addition, the SAP Sybase IQ main and shared temporary stores must not be I/O bound.

DQP uses the available memory and CPU resources of all nodes of the logical server. In general, the more nodes and resources are available, the better the query performance. There is an upper bound, based on the number of work units. If there are not enough work units to pass to all the available CPUs in the grid, only a subset of the CPUs will be used. The current workload of the nodes in the logical server will obviously affect performance.

Allocating more memory to temp cache promotes hash-based algorithms that are more likely to scale. A large temp cache is more important than a large main cache for DQP. I/O bandwidth of the shared temporary store, which is used to assign work and transmit intermediate results, is critical for the performance of a distributed query so if your storage layer offers tiered performance characteristics, placing IQ_SHARED_TEMP on the fastest storage will yield the best results.

This may seem obvious, but all distributed fragments must complete processing before the final result set can be generated and returned to the requesting application. So it should be noted that the “slowest performing fragment” will limit overall performance of the query. In addition, although queries are being distributed and load balanced automatically, it is still a good idea to load balance connections across the grid in order to spread the more intensive leader node responsibilities across all the nodes of the grid.

Queries Highly Likely to Benefit from DQP

DQP is intended for PlexQ grid environments that are heavily report intensive. Load performance is not affected by the DQP option, although loads can be parallelized by configuring multiple PlexQ writer nodes. Also, DQP will operate best when memory and CPU resources are balanced across the PlexQ grid.

Certain types of queries will scale better than others. Queries that are likely to distribute well have the following attributes:

• Compute-intensive column scans, such as LIKE conditions.
• Complex queries involving aggregation, expensive expressions, and numeric data types.
• Queries comprised of query fragments that reduce the size of intermediate or final results. An example of this is a chain of hash joins with a “group by hash” at the top.
• Low cardinality data often uses hash-based processing, which is more likely to scale. This occurs with star schemas, which are characterized by a large fact table with low-cardinality dimension tables.
• If you have medium cardinality data, you may be able to tune database options, and allocate more memory to temp cache, to bias the query optimizer to choose more hash based algorithms.

Queries Generally Unlikely to Benefit from DQP

As discussed earlier, certain types of queries inherently do not scale well and the optimizer may decide not to distribute them at all because they will probably perform best on a single node. Characteristics of these types of queries include:

• Queries that return many rows, such that returning rows is a large part of the query execution time. Note that producing rows out the “top” of a query is a serial operation that cannot be distributed.
• Small queries: queries less than 2 seconds in duration are unlikely to benefit from DQP; between 2 and 10 seconds are less likely to benefit; greater than 10 seconds are more likely to benefit.
• Queries with many fragments. If there are many fragments, this usually means that sorts are involved. This can lead to less scalability because sorting large amounts of data uses disk storage in the IQ_SHARED_TEMP DBSpace to perform the sort. This is another reason that the shared temporary DBSpace should be placed on the fastest storage possible.
• Joining high cardinality, large tables with each other will lead to merge joins. These do not scale as well as hash joins. Hash partitioned tables can mitigate this situation.
What You Can Do to Influence DQP Scalability

There are various server and database operations that affect parallelism and performance of a query:

- **Max_Query_Parallelism**: this database option sets an upper bound which limits how parallel the optimizer will permit query operators, such as joins, GROUP BY and ORDER BY. The default value is 64. Systems with more than 64 CPU cores often benefit from a large value — up to the total number of CPU cores on the system to a maximum of 512.

- **Force_No_Scroll_Cursors**: if you do not need backwards scrolling cursors, set this database option to ‘on’ to reduce temporary storage requirements.

- **Max_IQ_Threads_Per_Connection**: controls the number of threads for each connection. With large systems, you may see some performance benefit by increasing this value.

- **Max_IQ_Threads_Per_T eam**: controls the number of threads allocated to perform a single operation (such as a LIKE predicate on a column). With large systems, you may see some performance benefit by increasing this value.

- **Max_Partitioned_Hash_MB**: sets an upper bound, expressed in megabytes, on the amount of temporary space that the optimizer can assume will be available for hash-partitioned, hash-based query operators. The default is 0, which means unlimited. This option takes the place of the old “MAX_HASH_ROWS” option.

- **-iqgovern**: this server option specified the number of concurrent queries on a particular server. By specifying the -iqgovern switch, you can help IQ maintain throughput by giving queries adequate resources to commit quickly. The default value is \((2 \times \text{number of CPUs}) + 10\). For sites with large numbers of active connections, you might want to set this value lower.

- **-iqt**: this server option sets the temp cache size. Temp cache is used by both the local and shared temporary stores. DQP must utilize IQ_SHARED_TEMP in order to do its processing, and therefore requires adequate temp cache. You may want to allocate more memory to it than main cache for DQP workloads.

- **MPX_Work_Unit_Timeout**: when a worker node does not complete processing of its query fragment within the mpx_work_unit_timeout value, the work is passed back to the leader to retry. If you find that timeouts are occurring and adversely affecting the performance of DQP, you can increase the timeout value to allow a worker to complete. Generally, though, you are unlikely to hit a timeout issue unless you have some other underlying problem.

Sizing Shared Temporary Storage

An adequate amount of shared temporary space on fast storage hardware is critical for the performance of distributed queries. While it is difficult to calculate in advance how much shared temporary storage you will need for a distributed query, there are some trends that have been observed:

- Use of shared temporary space can vary widely among nodes in the PlexQ grid as they are executing a distributed query.
- The amount of shared temporary space used does not correlate with the scalability of the query. Queries that do not scale well may use as much or more shared temporary space as queries that do scale well.
- Queries that use more temporary cache/space when running on a single node will tend to use more shared temporary space when running distributed, but there is not an obvious multiplier that can be derived.
- Maximum amount of shared temporary space used across the PlexQ grid stays constant regardless of the number of nodes executing a particular distributed query.
- Amount of shared temporary space required on a node increases with the number of concurrent users executing the same distributed query. In other words, a higher workload requires more shared temporary storage.

Make sure that you have available storage to add to the shared temporary store if you find that it is not sized properly. You can add space dynamically without stopping the IQ server.
DQP SINGLE QUERY WORKLOAD TEST RESULTS

The range in performance of a distributed query varies significantly depending on the nature of the query, and the configuration and workload of the SAP Sybase IQ PlexQ grid it is executed on. The following results are the best achieved so far in a controlled, internal test environment.

These tests (a single large query initiated by a single client) were run on a SAP Sybase IQ PlexQ grid with the following configuration:

- **Dell Blade M1000E, Power Edge Enclosure**
  - 16 x M610 Blade Server; 56XX Processors (224-8593)
  - 2 x quad-core (Intel XeonE5620 2.4Ghz)
  - 48GB Memory
  - 2 x 300GB SAS Drives (Raid)
  - Dual-Channel 8Gbps Fibre HBA
  - Dual-Port 10GbE Network Card
  - 2 x Fiber Switch
    - Brocade M5424 FC8 Switch+AG,24 ports
  - 2 x 10GB Network switch
    - Cisco Catalyst 3130G, Gigabit Ethernet (223-5382)

- **10GB Private Network NFS Server**
  - Dell R710
  - quad-core
  - 24GB Memory
  - 8 x 1TB Near-Line SAS Drives

- **Storage**
  - 6 x PAC Storage 12-Bay 4Gb Dual RAID Controllers w/12 x 300GB 15K SAS Drives
  - 6 x PAC Storage 12-Bay EBOD (Expansion Shelves) w/12 x 300GB 15k SAS Drives
  - RAID-0 striping with LUN stripe size = 64KB

Each test on the following pages show the query plan of the particular query from the leader node, and a bar chart showing performance scaling from 1 to 8 server nodes. The name of the query has no particular significance, other than to uniquely identify it. In the query plans, note the “three bar” annotations indicating distribution of query processing.
Query_A

Figure 9 – Scaling Query_A from One to Eight PlexQ Nodes
Query_B

Figure 10 – Scaling Query_B from One to Eight PlexQ Nodes
SUMMARY AND CONCLUSIONS

This document has given you an overview of PlexQ, an exciting new set of capabilities in SAP Sybase IQ which includes Distributed Query Processing to enable a high performance, resource efficient, and operationally simple platform. DQP has been designed to take advantage of the CPU power of a SAP Sybase IQ PlexQ grid to scale the performance of large and complex CPU-bound queries. DQP can dramatically improve the performance of a query by breaking it up and distributing the pieces for concurrent execution across multiple SAP Sybase IQ servers. This new capability advances the SAP Sybase IQ platform to a “shared everything MPP” architecture that maximizes use of distributed resources to drive optimum query performance and resource utilization. The new data affinity feature introduced in SAP Sybase IQ 16 enhances performance even further by reducing I/O contention. Faster answers to time-critical business questions give organizations the edge in today’s increasingly complex and competitive world.