THOR™
The Hitachi Object Relational Database System

White Paper

Introduction

Purpose

This White Paper is designed to provide a broad technical overview of THOR™ – The Hitachi Object Relational Database System.

Audience

The audience for this White Paper is managers, executives, system administrators, and decision-makers with responsibility for large-scale DBMS systems.

Future White Papers

Forthcoming additional White Papers will discuss the technical details of THOR and its administration capabilities and tools in greater depth.

Executive Summary

We have extracted all the major ideas in this paper into an Executive Summary that begins on the next page. The summary consists of all the shaded paragraphs throughout the paper.

Contents of this White Paper

In the rest of this paper, we detail how THOR's hardware and software architecture combine to provide relief from the growth and management concerns currently facing DBMS managers and administrators.

Executive Overview

This Executive Overview consists of selected key paragraphs from throughout this paper.

Data processing requirements will continue to grow for the foreseeable future, yet staff and financial resources for DBMS applications and support will continue to be stretched thin. Managers are awash in data that they need to use in a variety of ways to maintain their competitive advantage. DBMS systems and tools must quickly evolve to manage the forthcoming data explosion.

The obvious solution to the problems facing today's DBMS managers is to use a central “industrial strength” data storage facility such as that provided by a relational DBMS (RDBMS), but one which extends the relational model to incorporate additional data types and operators. This is precisely what THOR has implemented in its Object Relational Database (ORDBMS) model.

THOR’s unique ORDBMS design addresses all the issues discussed to this point. But no matter how elegantly designed or how highly optimized for speed, a solution isn't a solution if it doesn't meet other basic criteria. Specifically, DBMS managers and administrators need a solution that is easy to administer, scaleable, extensible, and open.

Only one database system meets all these criteria — THOR

Because of THOR's unique methods of data distribution and parallel processing, THOR is able to dispense with many time-consuming administrative tasks that other systems require. THOR also implements a variety of features to ensure RAS.

THOR's shared-nothing approach, combined with its unique data-driven implementation, provides processing that is more inherently parallel and easier to distribute among processors than current implementations.

THOR was designed and implemented using object-oriented (OO) design tools and techniques, and is programmed to understand relational operators within this context.

Object capability allows developers not only to create new data types, but also to create new predicates, such as SIMILAR and PARTIAL, for the new data type FINGERPRINT. These predicates are evaluated by invoking calculation functions which the developer supplies, and which reside within the database just as EQUAL functions do today for each of the data types currently supported by SQL.

Another interesting example of how THOR supports new data types is the World Wide Web. Typical web sites offer several types of data, including text, pictures, movies, and sounds. THOR can support this mixed environment, especially when the requirement is for ad-hoc searches and queries.

By providing support for open interfaces, THOR preserves your investment in standard tools and applications. In fact, THOR SQL objectManager supports both standard SQL, through open interfaces such as the ODBC API, and the SYBASE Open Client/Server APIs, including Transact-SQL and DB/Net libraries.

The THOR MPP Data Server™ platform consists of four to 288 processing nodes configured in a shared-nothing configuration. THOR was designed from the ground up to be the first MPP solution that is both massively parallel and inherently object oriented. Of all MPP implementations, only THOR's SQL objectManager is truly object-oriented: all data is encapsulated with the behaviors or functions needed to operate on the data.

The software design of THOR is based on a data-driven model of execution, implemented using object oriented technology. Data-driven processing and object orientation combine to yield a product which is truly unique today, but which will be the paradigm for the next generation of implementations by other vendors.

Data-driven processing finds multiple operations that can be undertaken concurrently within the evaluation of one or more expression. Data-driven processing is inherently parallel, and overcomes the limitations of procedural programming, in which each statement must complete execution before the next statement begins. This model lets THOR process and evaluate many queries at the same time, further increasing performance.

THOR's unique method of distributing query initiation tasks among all nodes in the system, coupled with its data-driven and object-oriented approach to query processing at the node level, means that a large number of queries can be processed in parallel (inter-query parallelism). It also means that a single query can be processed in parallel on all nodes at once (intra-query parallelism.) In this way, THOR fully maximizes the parallel capabilities of its hardware and software system.

In summary, THOR was designed to address and resolve some of the most pressing issues facing database managers and administrators today. As you continue your reading and research in this important area of your business, we hope you will compare THOR to every other system you review. We are confident that THOR will come out on top.

Problem/Solution Statement

The Problem

Data processing requirements will continue to grow for the foreseeable future, yet staff and financial resources for DBMS applications and support will continue to be stretched thin. Managers are awash in data that they need to use in a variety of ways to maintain their competitive advantage. DBMS systems and tools must quickly evolve to manage the forthcoming data explosion.

The Solution

The obvious solution to the problems facing today's DBMS managers is to use a central “industrial strength” data storage facility such as that provided by a relational DBMS (RDBMS), but one which extends the relational model to incorporate additional data types and operators. This is precisely what THOR has implemented in its Object Relational Database (ORDBMS) model.

The Major Issues

THOR™ addresses four major issues currently facing DBMS managers and administrators:

- Overcoming performance barriers
- Controlling costs
- Supporting new data types such as pictures, movies, and sounds
- Moving toward data warehouse implementations

Overcoming performance barriers

Current OLTP implementations are reaching performance barriers. Systems cannot scale beyond a certain point, and performance barriers are being reached.

Controlling costs

Achieving maximum value for money spent is critical. Budgets are not growing as fast as performance requirements. As data processing requirements expand, systems need to scale appropriately.

Supporting new data types

New and more complex data types are becoming increasingly important in data processing environments. Managers need to query and report on non-traditional data using the same management tools available to work with traditional data types.

The changing data processing trend

The competitive power of complex query applications is no longer in doubt. Though they go by many names, including DSS (Decision Support System), Data Warehousing, Data Marts, OLAP (Online Application Processing), Data Mining, etc., these applications are rapidly becoming indispensable to any world-class enterprise. Data sets are growing significantly, requiring more processing power than many current OLTP implementations.

OLTP and DSS/OLAP differences

Introduction

There are three fundamental differences between OLTP and DSS/OLAP systems:

- Some data warehouses are far larger than the databases common in OLTP.
- OLAP consists of queries that are more complex than OLTP queries.
- Not all forms of data fit neatly into the SQL tables common in OLTP applications.

Larger databases

Some data warehouses are far larger than the databases common in OLTP, and the data to be analyzed are growing faster than processor power is increasing. Symmetric Multi-Processing (SMP) systems that easily handle large OLTP applications cannot scale fast enough to grow as data warehouse applications will, and are reaching performance limits.

More complex queries

OLAP consists of queries that are more complex than OLTP queries. Complex queries are much harder to parallelize than the simple atomic transactions of OLTP. Parallel systems engineered for OLTP and SMP systems can falter under the load imposed by complex queries.

Non-traditional data types and operators

Not all forms of data fit neatly into the SQL tables common in OLTP applications. Today’s vastly increased system capability has made possible a broad range of business applications. Data warehouses must increasingly house data that doesn't fit the traditional models of numbers or text, such as: time series data in the finance industry; geographic data in many industries; and pictures, images, video and sound in Intranet and Internet applications.

The limitations of the relational model have become a significant problem for a new generation of applications. Work is underway in the SQL language standards community to address these market requirements in the SQL3 standard, but this will not be ratified before 1998 at the earliest. Customers are demanding relief before then.

THOR’s Object-Relation model meets the criteria for a real solution

Introduction

THOR’s unique ORDBMS design addresses all the issues discussed to this point. But no matter how elegantly designed or how highly optimized for speed, a solution isn't a solution if it doesn't meet other basic criteria. Specifically, DBMS managers and administrators need a solution that is easy to administer, scaleable, extensible, and open.

Easy to administer

THOR reduces the system administrator's burdens of tuning the databases, optimizing queries, and partitioning distributed data, performing these tasks with significantly less manual intervention than current systems require. THOR also incorporates a number of RAS (Reliability, Availability, and Serviceability) features to ensure minimal down time and rapid recovery from system problems.

Scaleable

THOR lets you "rightsize" your hardware as your processing requirements grow, increasing processing speed almost linearly as new processors are added.

Extensible

THOR has an object-oriented foundation, making it easy to support new data types and data operators. At the same time, THOR works with your existing SQL-based RDBMS, enabling you to build object support on top of (rather than in place of) your current implementation.

Open

THOR uses existing tools and applications and leverages current RDBMS investment to the greatest extent possible.

Only one database system meets all these criteria —

THOR

THOR Tower configuration

THOR's Turnkey Solution

Introduction

THOR™ is a complete turnkey system, comprising:

- **THOR MPP Data Server™ hardware –** simple hardware optimized to provide a cost-effective platform for the SQL objectManager software
- **THOR SQL objectManager™ software** – an object relational SQL implementation designed to run on parallel shared-nothing hardware

The following sections briefly describe how THOR satisfies the criteria of being easy to administer, scaleable, extensible, and open.

THOR is easy to administer

Introduction

Because of THOR's unique methods of data distribution and parallel processing, THOR is able to dispense with many time-consuming administrative tasks that other systems require. THOR also implements a variety of features to ensure RAS.

Automatic data partitioning

THOR SQL objectManager distributes and partitions data across nodes via a proprietary hashing algorithm. Hashing provides a uniform data distribution, which leads to very good load balancing. Administrators no longer have to consider partition size and location issues as their databases grow in size.

Automatic query optimization

THOR uses a proprietary, highly tuned, cost-based query optimizer to speed up the processing of SQL queries. Administrators and programmers no longer have to tweak, tune, and test the majority of their queries to enhance performance.

Extensive RAS features

THOR incorporates a number of hardware and software design elements to minimize the possibility of system crashes, and to enable rapid recovery from any system problems. These features are briefly listed below.

- Hardware
  - ◊ N+1 redundant power supplies
  - ◊ Environmental monitoring
  - ◊ Diagnostics suite
  - ◊ Front-accessible disk drives
  - ◊ ECC protected memory and memory bus
  - ◊ Flash upgradeable firmware

- Software
  - ◊ Data and log mirroring across nodes
  - ◊ Support for ACID transaction principles
  - ◊ Parallel backup and restore
  - ◊ Remote system monitoring and configuration

THOR is scaleable

Introduction

THOR's shared-nothing approach, combined with its unique data-driven implementation, provides processing that is more inherently parallel and easier to distribute among processors than current implementations.

Shared-nothing approach

A *shared-nothing approach* means that each processor node on the system has its own memory, hard disk, CPU, and communications connections. Shared-nothing systems can provide linear speed-up and scale-up for up to thousands of processors, because they don't have the resource bottlenecks of shared-memory and shared-disk systems.

Data-driven implementation

Data-driven implementation means that queries are divided into discrete tasks that are executing at the same time on multiple processors. The transaction is complete when all the tasks are complete and the results assembled in the proper order.

THOR is extensible

Introduction

THOR was designed and implemented using object-oriented (OO) design tools and techniques, and is programmed to understand relational operators within this context.

Native OO support *and* native relational support

As with any OO system, THOR's OO design incorporates classes, attributes, and instances. The OO design is in contrast to the existing major RDBMS systems, which are based on procedural programming models. Because of their procedural nature, they have often had to be extensively rewritten to support major enhancements, frequently resulting in implementations that were poorly designed or that suffered severe performance problems.

THOR is also designed to understand row and column data and their associated relational operators. THOR easily performs relational operations on both traditional data types and on new, complex data types.

BLOB support

THOR also implements Binary Large Object (BLOB) support for new data types such as video, audio, and bitmaps. The object programming model guarantees that THOR is extensible for new data types and for defining new operations on data.

Easy support for new data types

Let's consider a simple example of a data type extension to SQL. Suppose we want to create a database of fingerprints. For each fingerprint in the database, we want to store some traditional SQL type data about when and where it was collected, to whom it belongs, etc.

In addition, we want to store the fingerprints themselves as part of the database. Since fingerprint is not a legitimate data type in SQL, we will need to add one or more new data types. For simplicity, we'll assume that one new data type will suffice.

Adding predicates for the new data type

Object capability allows developers not only to create new data types, but also to create new predicates, such as SIMILAR and PARTIAL, for the new data type FINGERPRINT. These predicates are evaluated by invoking calculation functions which the developer supplies, and which reside within the database just as EQUAL functions do today for each of the data types currently supported by SQL.

Querying on the new data type

The newly defined predicates can be used as qualifiers within a WHERE clause in SQL for queries to the fingerprint database. This makes it possible, for example, to screen the database for suspects in a particular crime based on similarity of fingerprints on file to one found at the scene. One way such support might be implemented is shown in the following pseudo-SQL example.

```
SELECT IMAGE, NAME, AGE, SEX, HT, WT, DOB
AS SUSPECT
FROM   TAKENPRINTS
WHERE  SIMILAR (IMAGE, SAMPLE)
  AND  NOT PARTIAL(IMAGE)
…
```

Web pages and data type support

Another interesting example of how THOR supports new data types is the World Wide Web. Typical web sites offer several types of data, including text, pictures, movies, and sounds. THOR can support this mixed environment, especially when the requirement is for ad-hoc searches and queries.

For example, a retail site might let a customer look up a specific type of suit based on certain criteria, such as color, cost, and fabric. THOR can return a picture of the suit, a movie clip of a model presenting it, and voice-over of the features. THOR and the World Wide Web can be a very powerful combination.

THOR is open

Introduction

By providing support for open interfaces, THOR preserves your investment in standard tools and applications. In fact, THOR SQL objectManager supports both standard SQL, through open interfaces such as the ODBC API, and the SYBASE Open Client/Server APIs, including Transact-SQL and DB/Net libraries.

Standard SQL gateway support

Figure 1 illustrates the arrangement of the THOR MPP Data Server relative to other computing devices in the data warehouse environment. THOR's standard SQL gateway allows the THOR server to accept data from a variety of sources, and to act as either a data warehouse or a data mart.

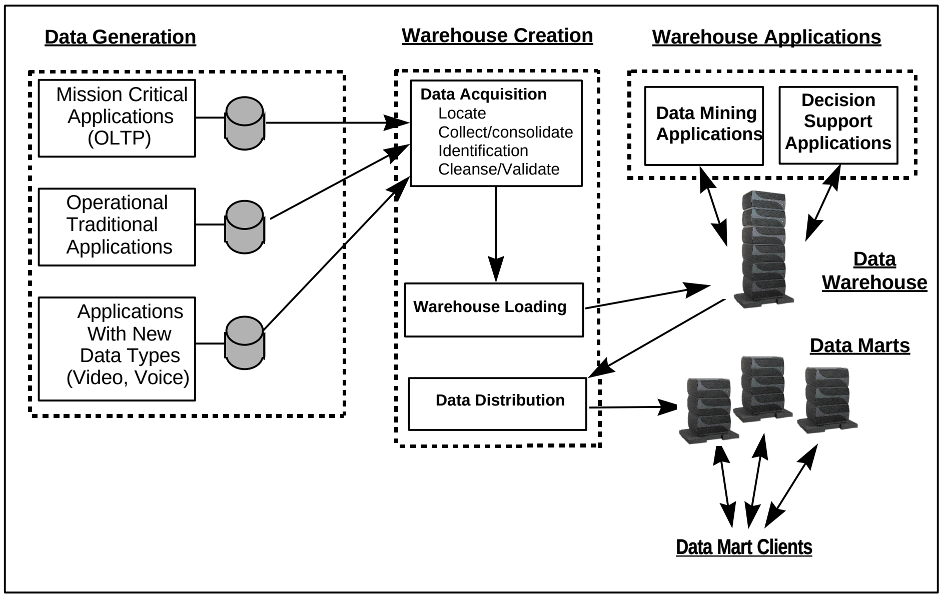

Figure 1. THOR in the data warehouse environment

Open architecture

Figure 2 illustrates THOR's open system architecture, and how it presents a single database view to the user.

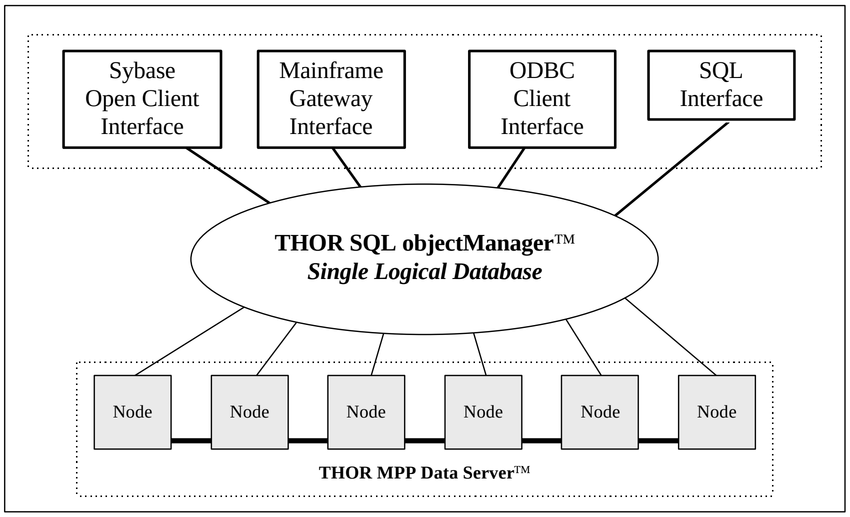

Figure 2. THOR's open system architecture

SYBASE Open Client and Transact-SQL support

THOR also supports SYBASE's Transact-SQL, an API that includes the SQL command set, and SYBASE's SQL Manager Open Server/Open Client APIs. This means that THOR supports many of the data types, SQL, stored procedures, and triggers compatible with SYBASE, one of the most widely used, open client-server platforms available today.

Figure 3 shows a possible THOR implementation, in which THOR fully participates with an existing SYBASE Open Client implementation.

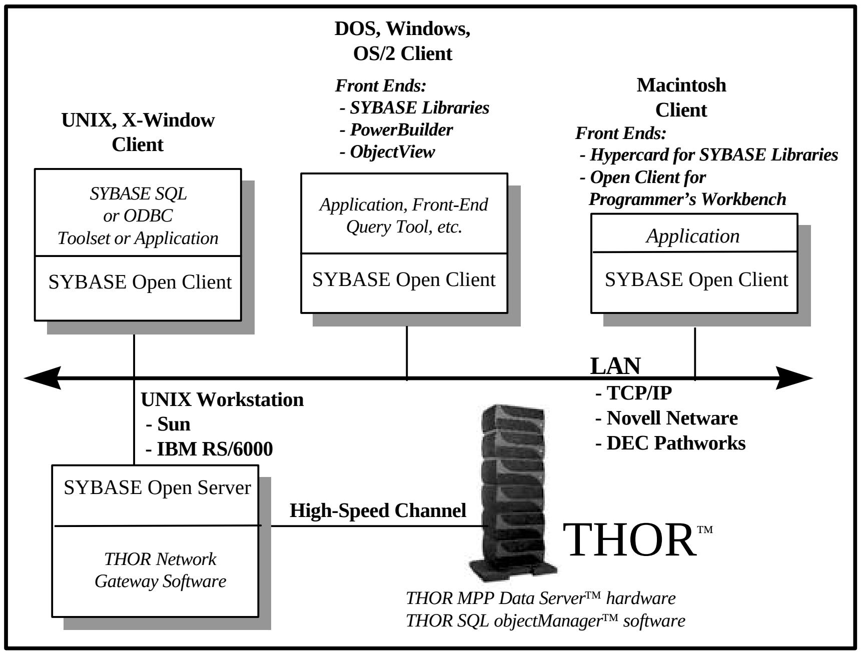

Figure 3. A THOR implementation using SYBASE Open Client

Plug-and-Play support

Application programmers who are using the latest application builder suites (e.g. PowerBuilder from SYBASE, Visual Basic from Microsoft, etc.) can create plug-and-play applications that interface with THOR's SQL architecture.

How THOR Achieves Maximum Performance

Introduction

The THOR MPP Data Server™ platform consists of four to 288 processing nodes configured in a shared-nothing configuration. THOR was designed from the ground up to be the first MPP solution that is both massively parallel and inherently object oriented. Of all MPP implementations, only THOR's SQL objectManager is truly object-oriented: all data is encapsulated with the behaviors or functions needed to operate on the data.

In this section of the White Paper, we describe how THOR's hardware and software models support maximum performance in a number of ways, including:

- Shared-nothing, MPP hardware configuration
- Unique toroidal mesh interconnect among processing nodes
- Uniform data distribution among nodes
- Underlying data-driven, object-oriented design
- Deadlock prevention via serialization instead of costly row and table locking
- Partitioned and pipelined parallism

Why THOR uses a shared-nothing, MPP hardware configuration

DSS/OLAP requires better scaling

There are significant advantages to an MPP architecture for processing the complex queries which characterize the DSS/OLAP market. These queries tend to be quite different from those in transaction processing. They often touch large amounts of data, are normally quite I/O intensive, and can run for very long times. They freely use SQL operators such as joins and aggregations that require processing large numbers of inter-row relationships, and thus they demand much more in the way of computing resources than typical OLTP queries. The result is that an architecture such as a shared-nothing MPP system, which isolates processors, scales much better than SMP for this workload.

Shared-nothing: definition

In a shared-nothing organization, the hardware is a set of processor nodes connected by a communication network. Each node contains its own CPU, memory, I/O bus, and disk storage. All communication between processors is switched through the communications network.

The shared-nothing organization can be compared to two other MPP hardware organizations:

- The **shared-disk model**, where each node has its own private memory, but can directly access all disks.
- The **shared-memory model**, where all nodes can directly access common memory and all disks.

Shared memory or shared disk systems don’t scale

At first sight, common access to shared data should make it easier to build a system, because any processor can have access to any data item that it wants. In fact, this is the main source of bottleneck problems. The intensive I/O activity in DSS/OLAP applications must compete with the CPU for memory bandwidth. In SMP and shared-disk or shared-memory MPP systems, as the number of CPUs increases, contention for resources such as memory becomes more intense, and performance suffers a severe impact.

Shared-nothing makes near-linear speedup and scale-up possible

Because of THOR’s shared-nothing architecture, the addition of more processing nodes does not lead to memory, disk, or CPU bottlenecks. In THOR's MPP implementation, each node has its own resources, so contention remains essentially constant no matter how many CPUs are added. In fact, the only contention in THOR is for communication bandwidth, and communication bandwidth increases as nodes are added, so never becomes a bottleneck.

This means that there is very little overhead relative to the size of a system; if system A has twice as many nodes as system B, system A will process queries and perform administrative tasks roughly twice as fast as system B. This capability ensures that managers can always scale a system appropriately, and add processing power simply by adding additional nodes.

THOR's unique toroidal mesh interconnect

Introduction

For maximum performance, SQL objectManager must have homogeneous connectivity among its nodes, with low latency, minimum interference among the traffic on the network, and minimum path length between source and destination nodes. THOR achieves this objective by using a unique toroidal interconnect, managed by a dedicated set of communication processors, one in each node.

Envisioning the toroidal mesh

The toroidal interconnect can be visualized as follows. First, imagine a rectangular connection grid with a processor node at each intersection, as shown in Figure 4 (a). Then connect the processing nodes on the right to those on the left with added network lines, forming a cylindrically connected system as shown in Figure 4 (b). Finally, envision connecting the nodes in the top and bottom planes as shown in Figure 4 (c). The nodes are connected as if they were laid out on the surface of a *torus*, or a doughnut shape.

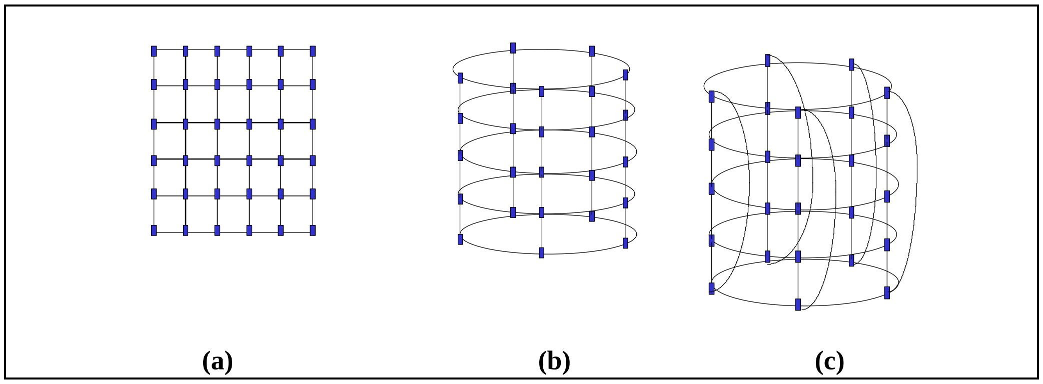

Figure 4. Envisioning THOR's Toroidal Mesh

How nodes interconnect

Each node interconnects with the other nodes in a “nearest neighbor” fashion. This connection employs special communication processors to offload the CPUs and is more cost-effective than other MPP systems that rely on multistage or crossbar switches or require the CPU to participate in the data routing. It also provides enhanced system availability through multiple path routing. (The communication processors are discussed more fully in “The node's communication processor” on page 17.)

The toroidal mesh topology has the important property that the array of processors is symmetric with respect to every node. Any function can be assigned to any node, and it will perform as well as if it had been assigned to any other node. This homogeneity is the most efficient communication design for optimal execution of the SQL objectManager software.

Uniform data distribution among nodes

Introduction

THOR uses processor *affinity*, combined with a proprietary *hashing algorithm*, to automatically and uniformly distribute database rows among processor nodes.

Why is uniform data distribution important?

For parallel processing to achieve optimum performance, table rows must be distributed evenly among a system’s processors. If four processors each contain 1000 rows, for example, and if the processors work in parallel, the time it takes to process all 4000 rows is roughly comparable to the time it takes to process 1000 rows. However, if one processor contains 2500 rows and the others contain 500 rows each, the time it takes to process all 4000 rows is roughly comparable to the time it takes to process **2500** rows, or all the rows on the overloaded processor. Uniform distribution, therefore, maximizes parallel processing performance for many operations.

Affinity: definition

A technique sometimes used to distribute data among nodes is called affinity. The term comes from the fact that each data item has affinity for a processor. Each processor is given a part of the database to manage, and all requests for that data are routed through that processor. THOR's shared-nothing system implements affinity; each processor accesses data and memory on its own disks.

Hashing: definition

To determine which row has affinity with which node, THOR uses a proprietary hashing algorithm. A hashing algorithm is simply a formula that generates a number indicating to which node a row should be assigned.

How THOR generates hash values

THOR's hashing algorithm uses a large random number, in conjunction with the primary index of a row, to generate the row's hash value. Bits from the hash value are used to select the node where the row resides.

To ensure even data distribution, the THOR SQL objectManager requires that the primary index for a table be unique. If a table has no primary unique index, THOR generates a synthetic hash value that ensures "round-robin" distribution of data to nodes.

Data-driven, object-oriented design

Introduction

The software design of THOR is based on a data-driven model of execution, implemented using object oriented technology. Data-driven processing and object orientation combine to yield a product which is truly unique today, but which will be the paradigm for the next generation of implementations by other vendors.

A brief description of data-driven processing

Data-driven processing finds multiple operations that can be undertaken concurrently within the evaluation of one or more expression. Data-driven processing is inherently parallel, and overcomes the limitations of procedural programming, in which each statement must complete execution before the next statement begins. This model lets THOR process and evaluate many queries at the same time, further increasing performance.

How queries are executed

As each user query comes in, it is assigned to the next available node in a "round robin" fashion. It is possible to do this only because the node array is symmetric with respect to each node, and because every node in the system can perform all the functions of the database system.

The assigned node is referred to as the initiating node or *query captain* for that query, and is responsible for:

- Parsing the query
- Optimizing the query
- Breaking the query down into *tokens*, or messages
- Obtaining serialization tags
- Accumulating the final results and performing the final aggregations
- Coordinating the transaction commit

Multiple nodes act as query captains for multiple queries in parallel, which leads to greatly improved performance.

Figure 5 shows a simplified schematic of THOR's toroidal mesh, illustrating how the query captain and the nodes communicate.

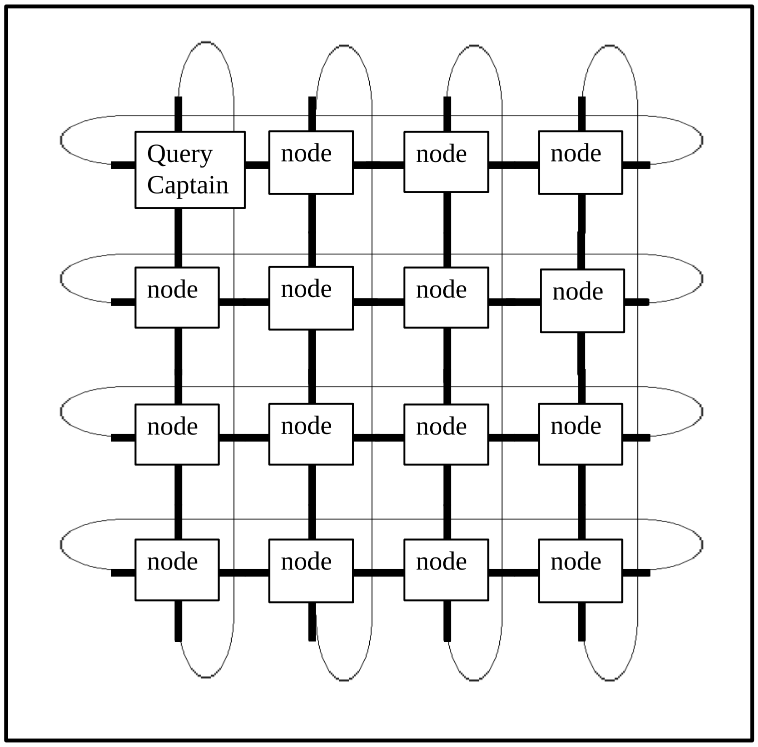

Figure 5. How nodes communicate during a query

As discussed in “Uniform data distribution among nodes” on page 11, each node is given a part of the database to manage, and all requests for that data are routed through that processor. This feature means that each node contains specific data, in the form of *row objects*.

Following is a simplified description of the procedure THOR uses to process and execute a query.

Step 1 – The query captain prepares and distributes the query

1a) At the initiating node, the query is parsed, analyzed, and optimized. This process defines a plan of execution for the database node software.

1b) The plan is then broadcast via *messages* to all nodes for execution by their row objects.

Step 2 – The row objects process the query

2a) Each row object determines whether it matches the query criteria and, if so, how it needs to compute and distribute data to successive objects or back to the query captain. The row objects communicate among themselves via messages to complete execution.

2b) As results for a set of rows on a node are available, the nodes distribute interim results to other row objects or back to the query captain.

Step 3 – The query captain assembles and returns results

3a) As results are pipelined in from the processing nodes, the query captain performs any final aggregations and calculations.

3b) When all results have been returned and all processing has been completed, the query captain returns the final results to the requester.

Summary

As the steps above illustrate, the only tasks that occur sequentially during a query are those performed by the query captain. All primary database processing takes place in parallel on all nodes at once.

THOR's unique method of distributing query initiation tasks among all nodes in the system, coupled with its data-driven and object-oriented approach to query processing at the node level, means that a large number of queries can be processed in parallel (inter-query parallelism). It also means that a single query can be processed in parallel on all nodes at once (intra-query parallelism.) In this way, THOR fully maximizes the parallel capabilities of its hardware and software system.

Inter- and intra-query parallel processing are discussed in more detail in "Partitioned and pipelined parallelism" on page 15.

Preventing deadlock

Introduction

In the past, parallel DBMS implementations have been severely affected by deadlock contention. Reducing deadlock contention has been a problem both for efforts to process multiple queries in parallel and for efforts to process a single query in parallel across multiple nodes.

Lock table can lead to bottle-necks

A common solution to this problem is for the software to mark pages as locked so that one processor can finish making its update before other processors are allowed to use the data. Unfortunately, the lock table is often centralized, and as more processors are added, the lock table becomes a bottleneck that limits system performance.

Serialization replaces the lock table

With data-driven as with any other execution model, it is necessary to ensure that operations take place in the correct order. In place of row-level or table-level locking, THOR uses a data-driven technique called *serialization* for most operations. The function of serialization is to make explicit which operations must precede other operations, and which can proceed at the same time.

Serialization tickets

Each operation that needs to be carried out (INSERT, UPDATE, SELECT, etc.) is assigned a *serialization ticket*. For each row object that requires a ticket, the row object is looked up and its serialization counters are incremented. Operations are then executed in the order that the tickets were issued. Once an operation has been issued tickets, the statement is serialized.

Tickets have two important properties:

- Tickets are just numbers, and thus are very fast and easy to issue.
- Tickets allow a row object to make a local decision about the order in which operations are executed.

Ticket generation is the only centralized function in THOR's software.

Read and write processes isolated

Each operation receives both a read and a write ticket, so that any number of reads can be carried out until a ticket for a write operation comes along. At that point, other read and write operations must wait only for the brief instant while the write operation is completing.

The major benefits of serialization

Serializing a statement has several important benefits:

- Many transactions that are deadlocked on other RDBMS systems are deadlock-free and restart-free on THOR.
- Even though the query may execute on many processors in parallel, all decisions about ordering operations can be made locally by the row objects. This minimizes inter-node communication, thus speeding up query processing.
- Serialization provides distributed concurrency control, and replaces the lock manager. There is no need for any other concurrency control mechanism.

Serialization and ACID principles

Serialization also assures that transactions meet the ACID principles for proper transaction processing. ACID stands for the following properties:

- *Atomicity* - The transaction is an atomic unit, consisting of a single collection of actions that is either completed in full (committed) or aborted in full (rolled back).
- *Consistency* - All integrity constraints of the database are fulfilled after the transaction has finished, thus leaving the database in a consistent state.
- *Isolation* - A transaction must be processed in apparent isolation. Data used by a transaction cannot change while the transaction is being processed. Similarly, other transactions can't use the results of the current transaction until it is committed or aborted.
- *Durability* - Once a transaction commits, its results must be durable, even in the case of software or hardware failures.

Partitioned and pipelined parallelism

Introduction

All of the features discussed above work in concert to enable two forms of parallel processing, partitioned and pipelined. Each of these is briefly described in this section.

Partitioned parallelism improves intra- and inter-query performance

Partitioned parallelism refers to a system’s ability to distribute a single query across multiple nodes at the same time. For example, imagine you have to go through the phone book to find the name of someone whose phone number you have. Obviously, this could take a very long time. Now imagine that the phone book is divided up into 100 sections of equal size, and 100 people are simultaneously looking for that name. The result will be found in approximately 1/100 the time. This is analogous to one of the primary ways THOR maximizes **intra-query** processing performance.

Partitioned parallelism also refers to a system’s ability to perform multiple queries on a node at the same time. Returning to our phone book example, this time imagine that your 100 people have to find two names instead of one. Instead of looking for only one name, each person can be looking for both names at the same time. Because both queries (“find a name”) are being executed in parallel, results will be returned in less time than if the queries were processed in serial, one after the other. This is analogous to one of the primary ways THOR maximizes **inter-query** processing performance.

Pipelined parallelism improves intra-query performance

Pipelined parallelism refers to a system’s ability to begin feeding results back as they are available. Continuing our phone book example, imagine this time that you have to produce a sorted list of all people whose phone number begins with 234. If you are the query captain (as described in “How queries are executed” on page 12), you are responsible for sorting the names your 100 workers find.

Instead of waiting for each of your workers to find all the matching names, you ask them to hand you their list of names in sorted batches of 20 at a time. Each time you receive another sorted 20 names, you inter-sort them with the other names you have already received and sorted. In this way, names are being sorted almost as fast as they are being found; there is no significant delay between finding all the names and producing a sorted list. This is analogous to another primary way THOR maximizes **intra-query** processing performance.

THOR's Architectural Configurations

Data Processing Nodes

Nodes

In Hitachi's THOR MPP Data Server™ architecture, the basic building block is a *Data Processing Node*, simply referred to as a node. In the THOR system, a node is a complete RISC-based computer system equipped with a processor and I/O subsystem that interconnects to other such nodes. The processors and adapters are all connected via a local PCI bus, and all are packaged together within a compact field-replaceable unit.

As noted earlier, the components have been selected to provide the maximum price/performance, with high volume, off-the-shelf parts used wherever possible. The system is designed to function in a normal office operating environment.

The configuration within each processor node is shown schematically in Figure 6.

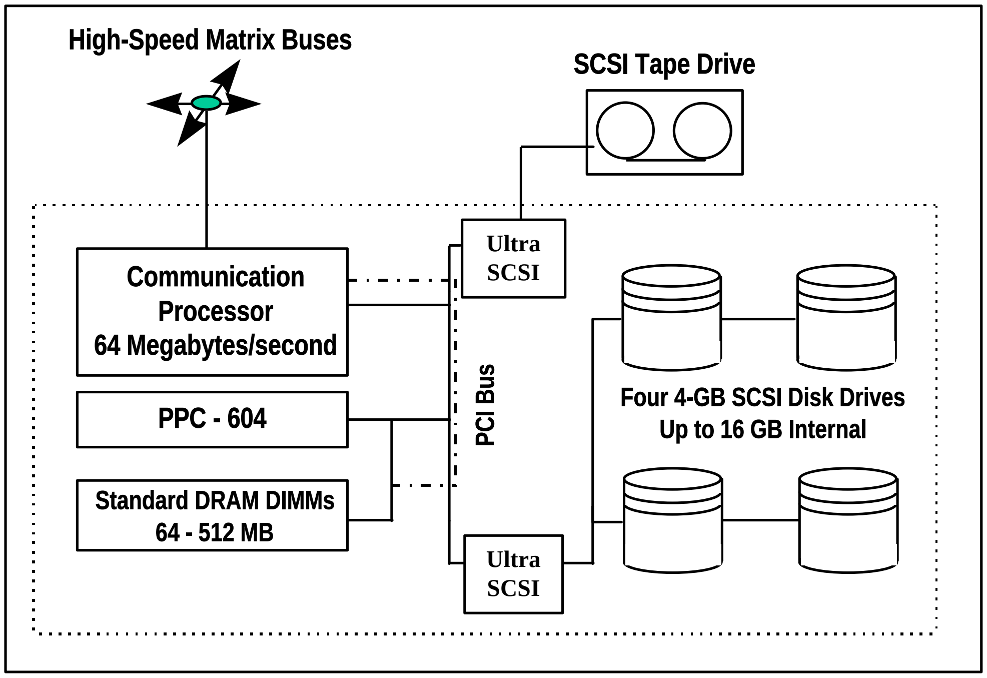
High-Speed Matrix Buses
SCSI Tape Drive
Ultra SCSI
Communication Processor 64 Megabytes/second
PPC - 604
Standard DRAM DIMMs 64 - 512 MB
PCI Bus
Four 4-GB SCSI Disk Drives Up to 16 GB Internal
Ultra SCSI

Figure 6. Processor node schematic

Two kinds of nodes

There are two kinds of processor nodes, distinguished mainly by the software present on them. Interface processor nodes (IPNs) connect THOR to the outside world through a high-speed channel attachment. Database processor nodes (DPNs) are repositories for the data and contain database query execution software.

Unlike some other systems, THOR IPNs may also act concurrently as DPNs. Figure 7 illustrates a schematic of a possible IPN and DPN configuration.

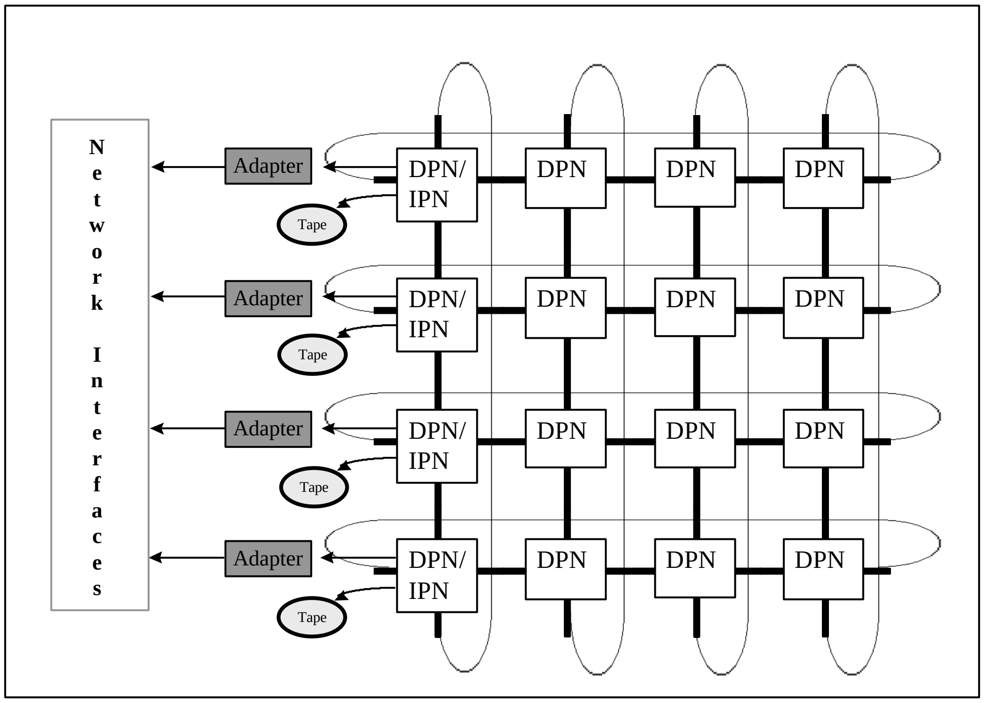

Figure 7. Schematic view of THOR's IPN and DPN configuration

The node's communication processor

As indicated earlier, the dedicated communications processor connects the node to the toroidal mesh backbone network. The communications processor has DMA access to the memory and supports four simultaneous bi-directional links, for a total sustained bandwidth of 64 Megabytes per node per second. The processor also functions as a router to significantly offload the main processor.

Modules

Four nodes = one module

Up to four nodes are packaged together into a *module*, which contains cabling attachments and N+1 redundant power supplies. The module is the smallest stand-alone unit of a THOR system; any THOR system must have enough module enclosures to contain all of its nodes. A module supports a sustained bandwidth of 256 Megabytes/second.

The database system and the operating system operate together to distribute user data, manage each node's devices, process queries, and provide communications. This means that all the components in the module operate at near linear scaleability as the nodes balance the workload for data processing. To increase the processing power, simply add more modules to the system and the workload capacity processing power will increase proportionally.

Towers

One to six modules = one tower

As shown in Figure 8, modules may be stacked into "towers" up to six modules high, and towers may be interconnected to expand a THOR system still further. A tower with six modules supports a sustained bandwidth of 1.5 Gigabytes/second. Although there is no architectural limit to the number of towers, the present hardware system has been designed to support up to twelve interconnected towers for a total of 288 nodes.

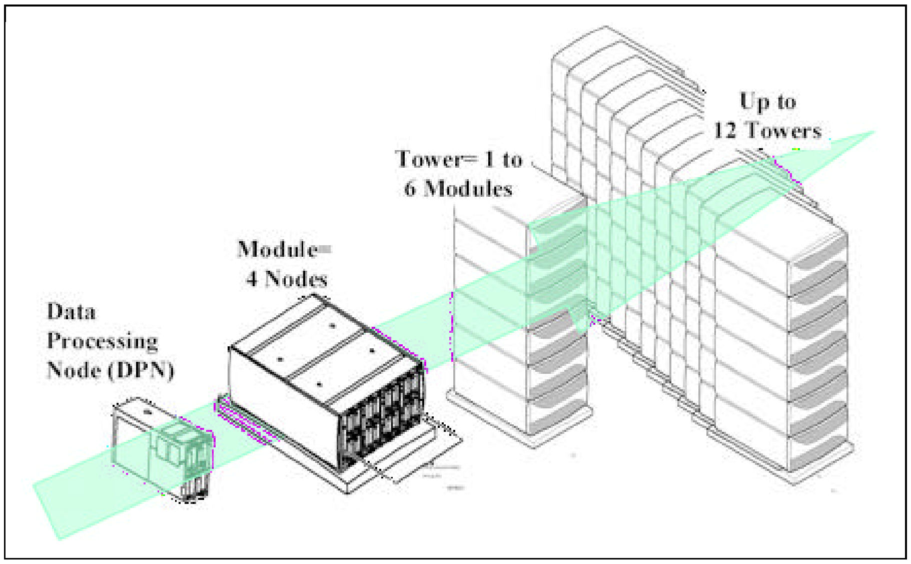

Figure 8. Data Processing Nodes, Modules and Towers

The Bottom Line

Introduction

In this section, we summarize how installing THOR can benefit your bottom line. All ideas here are based on information presented throughout this paper.

Ease of integration into your network

THOR can seamlessly integrate into an existing LAN system. This translates into maximum immediate benefits with maximum retention of current investment, and minimal network reconfiguration or data distribution.

Ease of administration

THOR provides a variety of tools and technologies that eliminates some administration tasks while speeding up most others. THOR also incorporates a number of features to ensure a high RAS standard. This leaves system administrators with more time to support users, improve reporting capabilities, and plan for future DBMS requirements—tasks worth more to your business and more to your bottom line.

ODBC/ SYBASE compatibility

THOR's full support for ODBC and SYBASE compatibility benefits your bottom line in several ways.

- Compatibility enables you to migrate existing applications with a minimum of rewriting.
- Compatibility lets programmers pick up where they left off in implementing applications, rather than making them start again in a new language.
- SYBASE compatibility ensures that any administrator with SYBASE experience already knows a good deal about how to administer THOR.
- You won't have to hire a system administrator who needs to learn a whole new system from scratch.

Near-linear scaleability

THOR's unique hardware and software design ensures that you can add processing power only as you need it, and get full value for your investment. Buying more than you need in anticipation of future needs is expensive, and keeps your capital tied up for long periods of time. With THOR, this expense is incurred only when your DBMS needs grow beyond your current capability, and only to the extent required.

In conclusion

In summary, THOR was designed to address and resolve some of the most pressing issues facing database managers and administrators today. As you continue your reading and research in this important area of your business, we hope you will compare THOR to every other system you review. We are confident that THOR will come out on top.

Who is Developing THOR, and Why?

Who

THOR™ is being developed by Hitachi Computer Products (America), Inc. (HICAM), part of the Hitachi family of computer and communications companies. Other companies in the computer and communications group include Hitachi America, Ltd., Hitachi Data Systems (HDS), and Hitachi Personal Computer (HiPC).

Why

Fortune 1000 customers have always looked to Hitachi for leading edge hardware and software solutions for other markets, such as large scale enterprise computers and data storage. Hitachi recognized that these same customers are facing a new kind of challenge in their data management strategies. These customers are awash in data. They have hundreds, maybe thousands of independent data storage locations scattered throughout enterprise networks. What they need is help turning their raw data into useful information.

Hitachi has chosen to participate in the VLDB market at this point because customers are unhappy with the rate at which their database needs are being served, and have asked for our help. Looking at the state of the database industry and its market, we agree with our customers that their ability to respond to their customers' needs, to become more competitive, and thus to grow, is currently constrained by the available technology. By drawing on its experience, Hitachi created a next-generation, object-relational database system that allows customers to extract information from enterprise database systems while maintaining current investments in hardware, software, and training.

Where Can I Learn More about THOR?

Phone

If you would like more information about THOR™, you can contact Hitachi by telephone at 1-800-588-THOR (1-800-588-8467) or 408-588-3300.

Fax

You can reach Hitachi by fax at any time. The number is 408-988-1279.

Internet

Additional information is also available on the World Wide Web at http://www.hicam.hitachi.com/thor.

Features and Specifications

The tables in this section describe THOR's hardware/software environment and basic features.

Table 1. THOR™ Hardware Specifications

HW Specifications	Module (four nodes)	Tower (24 nodes)
Model Number	HI64	HI512
Memory	256 MB - 2 GB, ECC protected	1.5-12 GB, ECC protected
Total disk space	67.2 GB	403.2 GB
Network gateway connections	up to 4 Fast-Wide SCSI-II	up to 24 Fast Wide SCSI-II
Processors	4 of each of the following: • Power PC 604 RISC Processor • PowerPC bus w/ 1MB Synchronous Cache RAM (SRAM) • 32-bit PCI Bus • Dual Ultra-Wide SCSI Buses • up to 512MB 60ns EDO DRAM w/ECC protection • Dedicated Communications Processor	24 of each of the following:
I/O	Node to Node communications: • Toroidal interconnect topology • Sustained bandwidth of 64 Megabytes/second per node ◊ Per module: 256 Megabytes/second total sustained bandwidth ◊ Per tower: 1.5 Gigabytes/second total sustained bandwidth External Peripherals: • One Fast-Wide SCSI II port per node Internal Peripherals: • One Fast-Wide SCSI II channel per node • Four hard disk drives per node Diagnostic Ports: • One RS-232 Serial Port per node • One Ethernet 10BaseT per node	

Table 2. Summary of THOR Features

SQL Supported	SQL and SYBASE TRANSACT-SQL – Supports stored procedures and triggers – Full set of system administration tools
Database integrity	Row-level concurrency control
Data types	Supports SYBASE SQL Server data types
Security control	Login and password – GRANT/REVOKE privileges – Views
Backup/recovery	Full transaction logging (logical-mirrored) – Transaction roll-back and two-phase commit – Backups (DUMP) on databases – Automatic recovery – RAID level 1
Networks supported	All networks supported by SYBASE Open Server – Ethernet (TCP/IP) – Novell NetWare LAN Workplace – Microsoft LANManager – IBM LAN Server
SYBASE Open Server Interface	Supports SYBASE Data Workbench, APT Workbench, Report Workbench – Supports SYBASE and Microsoft SQL Server-compatible 4GLs, gateways, and utilities

Table 3. THOR Technology Parallel Features

Product/ Feature	THOR SQL objectManager
Data partitioning	Hash
Indexing	Hash
Isolation	Yes
Parallel-Scan	Yes
Parallel-Sort	Yes
Parallel-Aggregate	Yes
Parallel-Join	Yes
Parallel-Recovery	Yes
Parallel-Insert	Yes
Parallel-Delete	Yes

Abbreviations Used in this White Paper

ACID - Atomicity, Consistency, Isolation, and Durability
BLOB – Binary Large Object
DBMS – Database Management System
DPN - Data Processing Node
DSS – Decision Support System
IPN - Interface Processing Node
MPP – Massively Parallel Processing
OLAP – Online Analytical Processing
OLTP – Online Transaction Processing
OO – Object-Oriented
RAS – Reliability, Availability, and Serviceability
RDBMS – Relational Database Management System
SMP – Symmetric Multi-Processing
SQL – Structured Query Language
THOR – The Hitachi Object Relational Database System
VLDB – Very Large Database