Clustrix

A New Approach: Clustrix Sierra Database Engine

The Sierra Clustered Database Engine, the technology at the heart of the Clustrix solution, is a shared-nothing environment that includes the Sierra Parallel Planner and the Sierra Distributed Execution Engine. Sierra provides an entirely new architectural approach to query resolution. It moves the query to the data, not the data to the query. Learn how this revolutionary database technology makes it possible to scale a single database across nodes, and still support massive concurrency and deliver high performance, full relational functionality, transactional consistency (ACID), and seamless deployment.

Clustrix has produced the first truly scalable fault tolerant Clustered Database System (CDS). A complete CDS supports the scalability and data model flexibility of a key-value store, while providing the robust relational functionality and ACID/immediate consistency and seamless deployment of a SQL system. Clustrix's CDS solution can handle queries from simple point selects and updates to complicated SQL joins and aggregates. It is optimized for highly transactional OLTP workloads and also works for OLAP queries. The Clustrix architecture can start small and expand seamlessly with business needs to arbitrary scale. Tables can range from 0 to billions of rows in size. Workloads can range from a few to hundreds of thousands of transactions per second. It can handle simple key / value operations to full ACID compliant transactional SQL. This is all powered by the Clustrix Sierra Database Engine (Sierra).

Bring the Query to the Data, Not the Data to the Query

The core of Sierra is the ability to execute one query with maximum parallelism and many simultaneous queries with maximum concurrency. In order to do that, a fundamental change was needed in query resolution. A traditional monolithic database cannot scale simply by bolting on an expandable storage layer. A distributed storage engine with a traditional planner and execution environment does not allow sufficient concurrency to scalea table to billions of rows and still obtain reasonable performance. A new approach is needed that encompasses the entire stack from the query compiler down to the storage engine.

The key observation to be made is that local queries can be satisfied with local locking, local data, and local cache. A query operating on local data need not talk to other nodes. Locks on the data structures can be very short lived. Operations on different bits of data can be completely independent and operate with perfect parallelism. The amount of total concurrency supported becomes a simple function on the number of independent data stores that contain that data. The magic then becomes the engine that ties these independent, high performance data stores into a global single-instance database. It is the Clustrix Sierra engine that makes this possible.

Sierra's most basic primitive is a compiled program called a query fragment. The query fragments are compiled all the way down to machine code and have a very rich set of operations that they can perform. They can read, insert, update a container, execute functions, modify control flow, format rows, perform synchronization, and send rows to query fragments on other nodes. These query fragments are run on the nodes that contain the data. The communication between the nodes consists of just the intermediate and result rows needed for the queries. Many, many query fragments can operate simultaneously across the cluster. Those query fragments may be different components of the same query or parts of different queries. The result is the same: massive concurrency across the cluster that scales with the number of nodes.

 © 2012 Clustrix, Inc. All Right Reserved

Sierra Overview

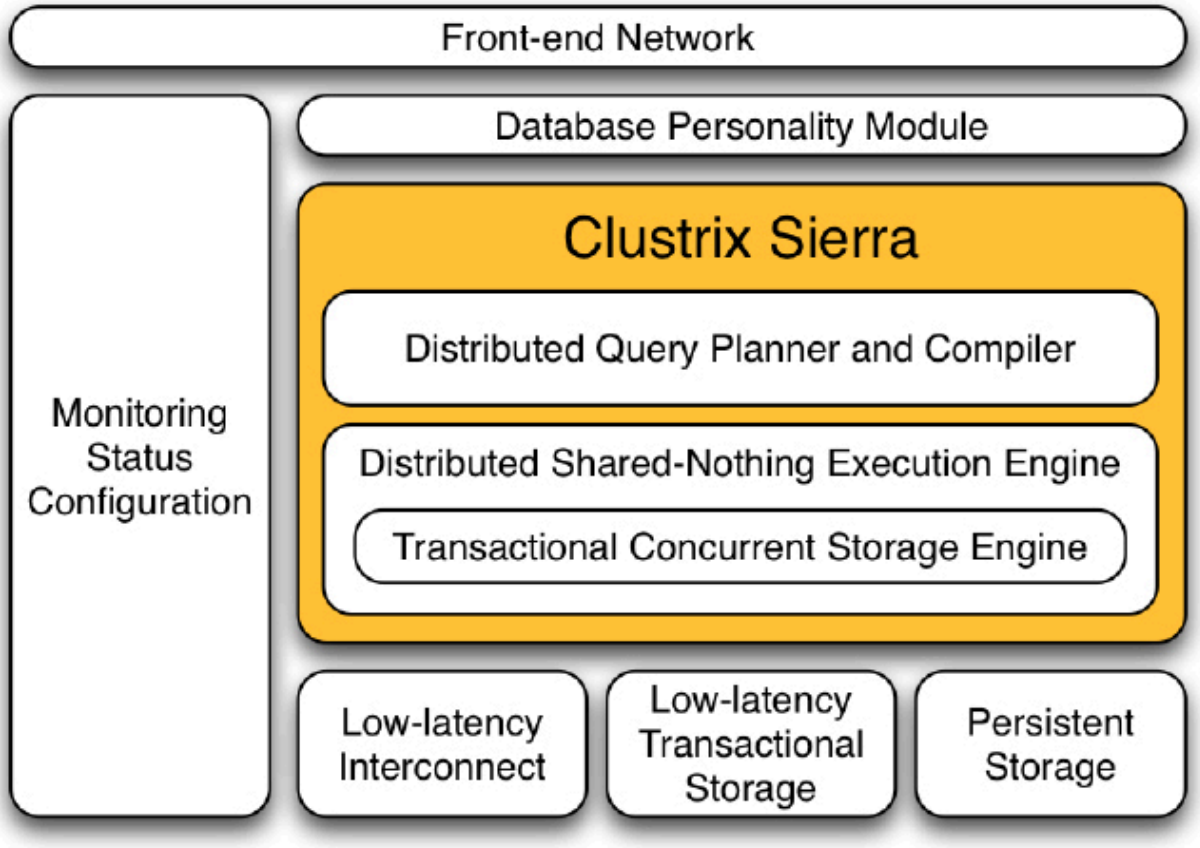

Figure 1: Block diagram

Sierra is a distributed computing environment that includes a distributed query compiler and a shared-nothing parallel execution environment with a transactional concurrent storage engine (See Figure 1). These different parts work together to be able to execute user queries with maximum efficiency, concurrency, and parallelism. Queries enter the system through the front-end network and are translated by the database personality module to an internal representation used by Sierra. Sierra then executes the queries in an efficient parallel fashion. Sierra uses persistent storage (SSDs) to store the data, low-latency transactional storage (NVRAM) to journal changes, and a low-latency interconnect (Infiniband) to communicate with other nodes in the cluster. Integrated at every level in the stack are extensive monitoring, status, and configuration facilities.

The best way to explain how Sierra works is to go through some example queries. This paper will go through a simple point select, a join, a select with order by and limit, and an insert, explaining what makes Sierra unique. This will show how the flexible components that make up Sierra enables it to resolve any query and do it with maximum concurrency.

© 2012 Clustrix, Inc. All Right Reserved

Data Layout

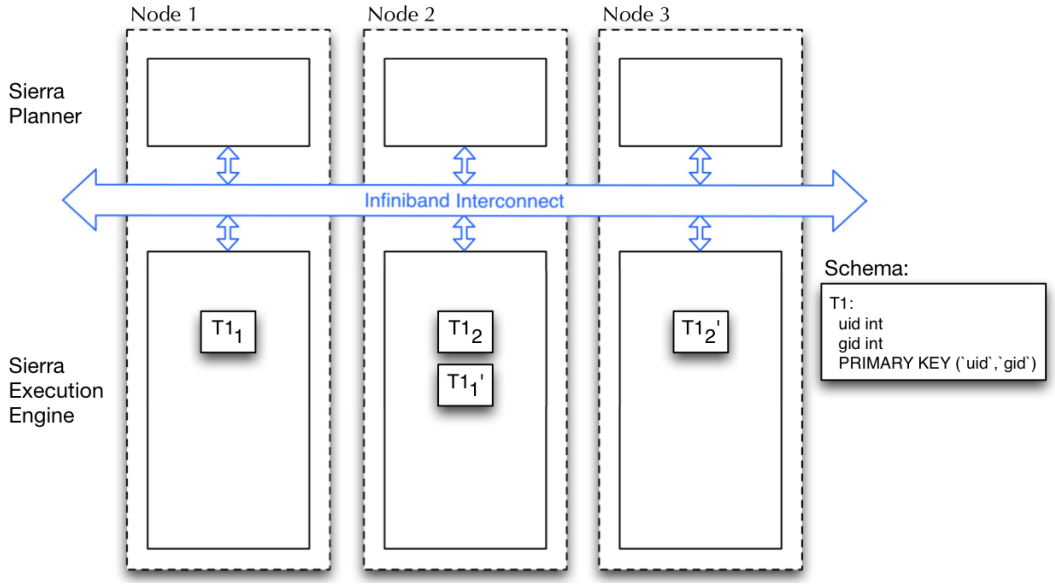

Figure 2: Data layout for a simple table

Figure 2 shows a physical representation of the data layout for a simple table, T1. Sierra partitions tables into objects called slices and those slices have replicas for data redundancy. In this example, table T1 has 2 slices: T11 and T12; additionally, each slice has a replica labeled T11' and T12' respectively. The slice and its replica contain identical data and can be used interchangeably by Sierra. There can be any number of slices per table, generally determined by table size. Sierra will automatically and transparently split slices when they get too large. The number of slices can be also be set by the user. The data placement of these slices and replicas throughout the cluster is dynamic. Slices can be moved while the cluster is online with no disruption to client queries. When a new node is added to the cluster, data is automatically moved there to rebalance the cluster. When a drive or a node fails, the slices that were contained on the failed device are automatically reconstructed using the remaining resources in the cluster. This data redundancy, automatic healing, along with a dual Infiniband interconnect, ensures there is no single point of failure in the cluster.

The data cache in Sierra is local to the node that contains the data. In this example, the slice T11 and its related cache live on node 1. Writes go to all replicas for redundancy (T11 on node 1 and T11' on node 2 in this example). Sierra treats one replica of a slice as the primary for reads (T11 here). Having a primary read replica ensures maximal cache utilization by avoiding caching the same data in multiple places in the cluster. Sierra can dynamically change which replica is primary based on dynamic load in the cluster. If you contrast this with a shared disk system that pulls the data to the query, you either have high latency for data movement around the cluster when queries are executed or you have to cache the data on the machine where the query is run. This can mean many copies of the same data in cache, greatly reducing cache efficiency.

The distribution of data among the slices is determined by a distribution function: dist(<key>). That distribution function can either be range or hash based. The number of components in a compound key contributing to the distribution can be selected. In the example in Figure 2, data can be distributed on (`uid`) or the combination (`uid`,`gid`).

© 2012 Clustrix, Inc. All Right Reserved

Point Select

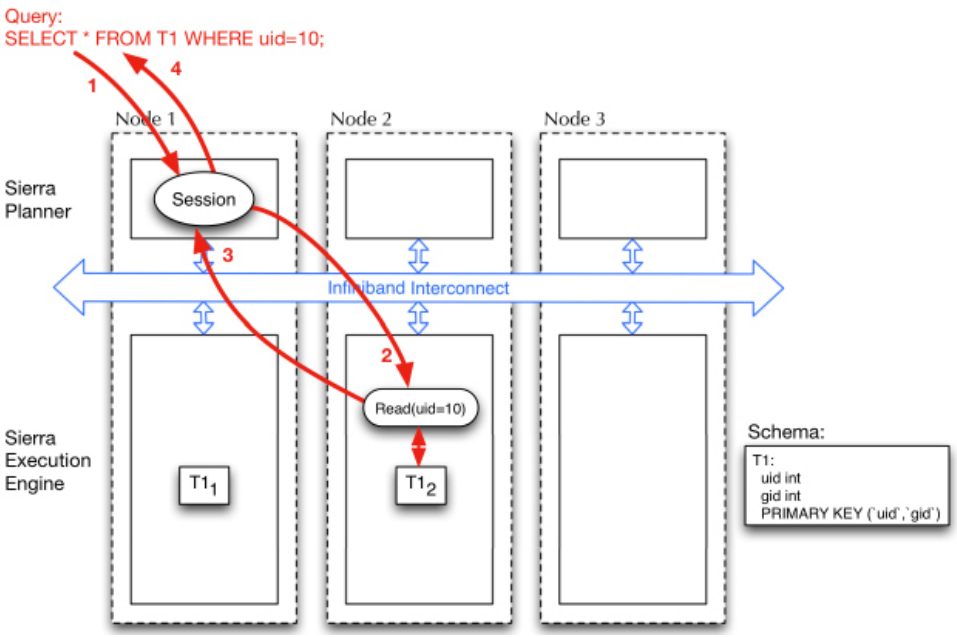

Figure 3: Point Select

Figure 3 describes the simplest of all queries in a database, the point select. For clarity, the replicas have been removed from the picture. In this example, a query comes into the client's session on node 1, the Sierra planner generates a plan, this plan is executed in the Sierra execution engine, and the results returned to the client. There is more going on here than is obvious at first.

The SQL query in step 1 is parsed by the session and turned into a query fragment by the Sierra Planner. Using the distribution function dist(uid=10), Sierra decides T12 is the slice that contains the relevant data. The query fragment, along with any necessary constants (uid=10) is what is sent to node 2 in step 2. The query fragment is represented in this diagram by the rounded rectangle. These query fragments can be cached with parameterized constants so they don't have to be generated every time. In this case, the query fragment does a container read and finds all rows matching the constraint uid=10 and returns the rows in step 3. Those rows are then sent back to the client in step 4.

A point select in Sierra involves no global table or row level locks. Sierra does this with MVCC or Multi-Version Concurrency Control to ensure consistent data is returned. Very short-lived local page-level locks protect the on- disk data structures themselves. Even those locks are scalable given that they are per-slice and the number of slices grows with the size of the table.

If you contrast this with a database that is distributed at the storage layer, the story is quite different. In that sort of architecture, the data would be pulled to the node that is running the query. There, the locking is global and necessarily higher latency. Global lock contention and drastically increased data movement severely limits the potential scalability of that system.

© 2012 Clustrix, Inc. All Right Reserved

Two Table Join

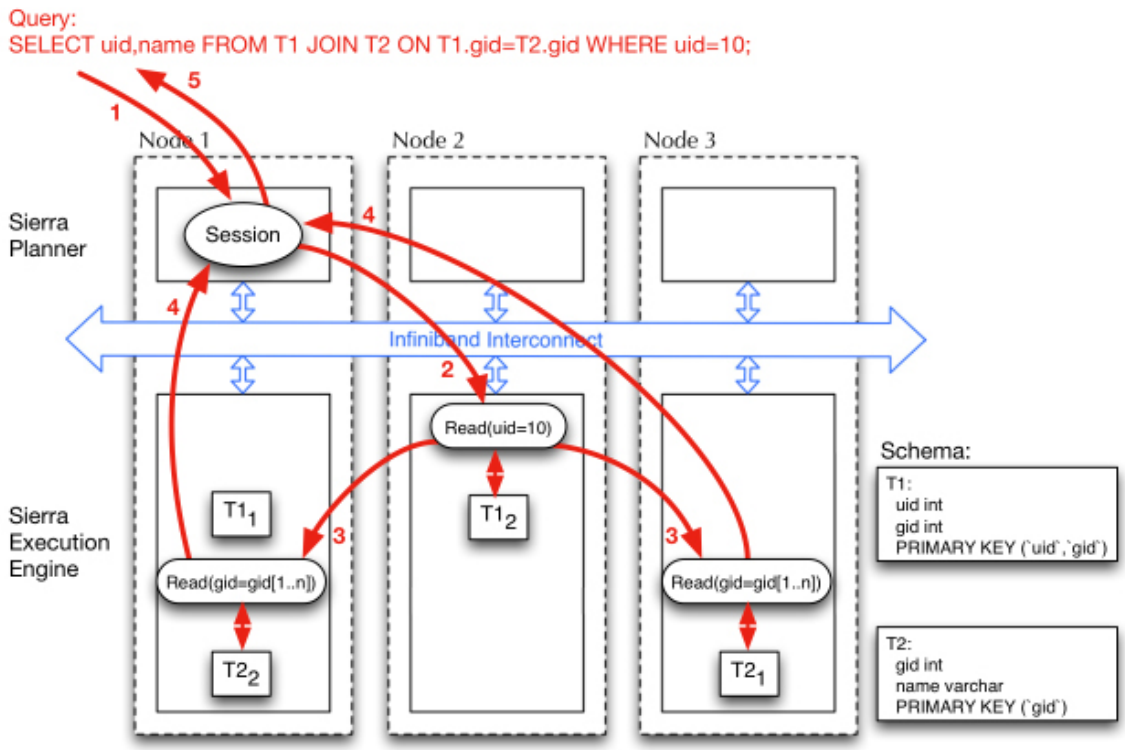

Figure 4: Table join with constraint

Figure 4 illustrates a two-table join with a constraint. Here like with the point select, the query comes in, gets compiled down to machine code query fragments, is routed around the cluster for query resolution, and the rows are sent back to the client. The plan generated in this case is more complicated. The Sierra planner has flexibility in join order, access path (which index it uses), general operation ordering, distribution, and parallelism. Sierra uses a variety of statistics to make intelligent decisions when making the plan. In this example, the generated plan is illustrated in Figure 5 as a dependency graph. The dependency graph is closer to the internal representation the Sierra planner uses when creating the plan. There, it's easier to see the dependencies, of course, but it also illustrates which data is transferred in each step. The numbered steps in the dependency graph correspond with the steps in the physical representation.

© 2012 Clustrix, Inc. All Right Reserved

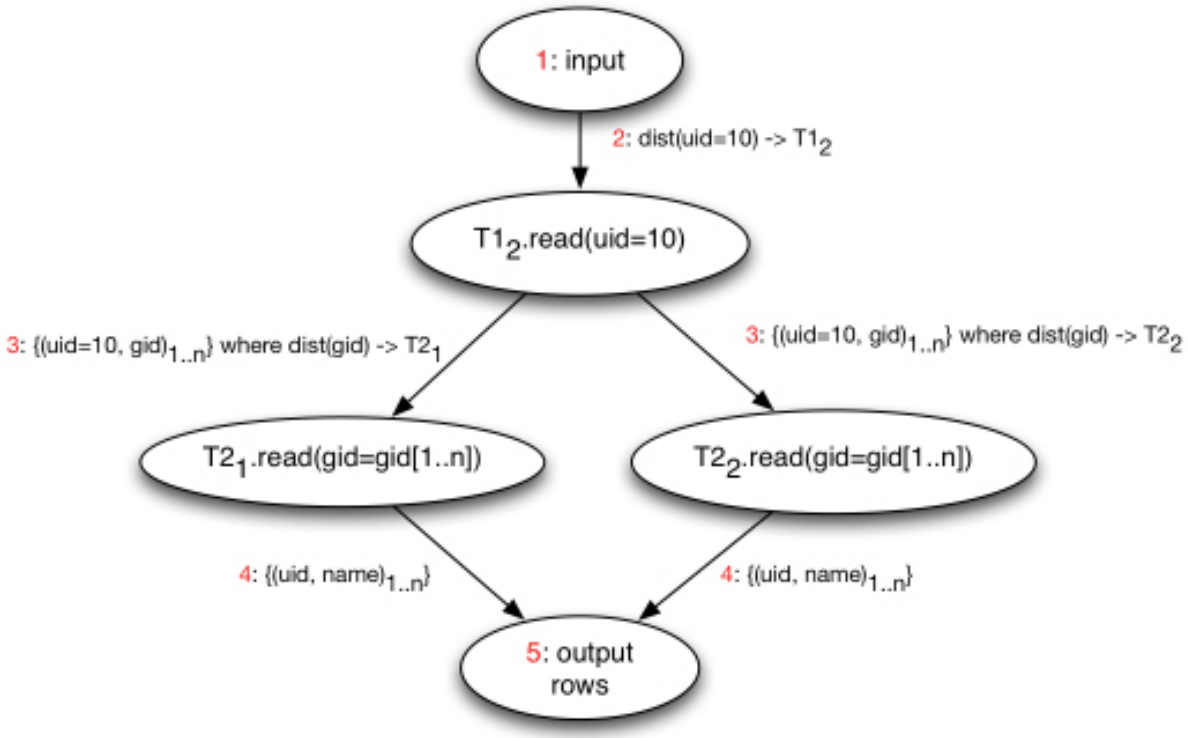

Figure 5: Dependency graph for two-table join

As before, the SQL query in step 1 is compiled to machine code. In step 2, the query fragment is sent to the slice determined by the distribution function dist(uid=10) of the first table of the join. There, it finds all rows that match the constraint uid=10 and forwards the rows along with the query fragments to the appropriate slices in step 3. Here, it uses the distribution function dist(gid) to decide where to forward each row retrieved from the slice T12. Slices T21 and T22 are read to find the rows that match the T1.gid=T2.gid constraint. The appropriate result rows are sent in step 4 to the session and the result is sent to the client in step 5. It's important to note that only the relevant columns of the necessary rows are sent at each stage. In this example, (uid,gid) is sent in step 3 and (uid,name) is sent in step 4.

In this join example, like the point select, there is no global locking, the data lookup for different slices happen in parallel, and there are no global points of contention. Given the scalable nature of the Sierra architecture, the tables T1 and T2 can have any number of rows and any number of slices. As the cluster expands, the available resources to store tables and execute queries also expand. Sierra truly represents full relational calculus capability at arbitrary scale.

© 2012 Clustrix, Inc. All Right Reserved

Select With Constraints, Order By, and Limit

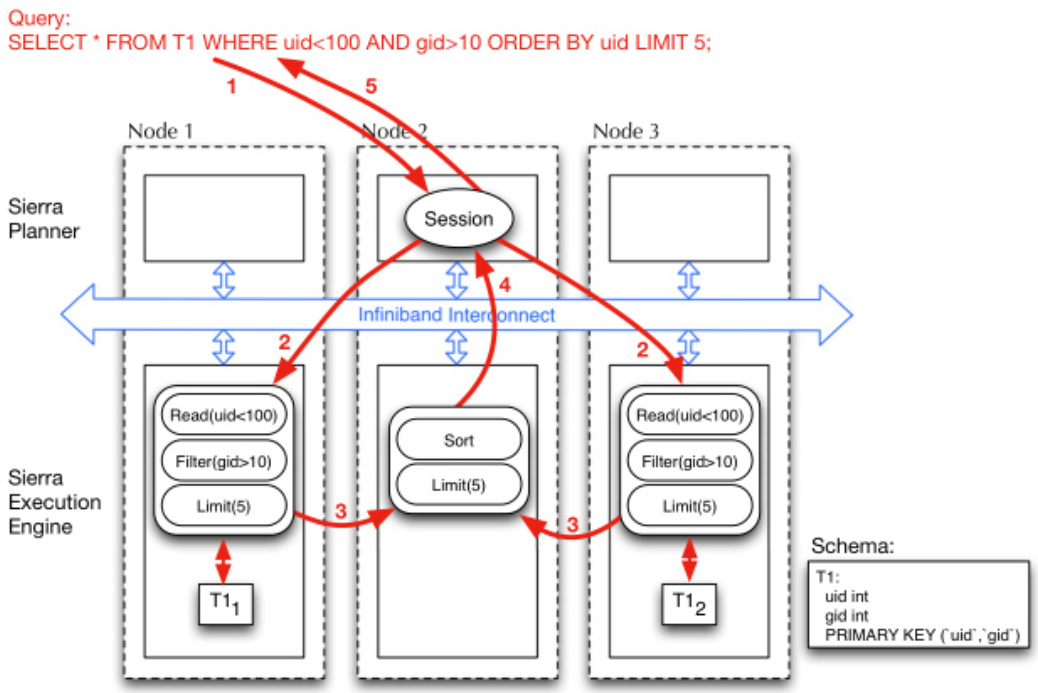

Figure 6: Group by with limit

Figure 6 shows a more complicated query that illustrates more of the optimizations available to Sierra. The query has two constraints plus a limit and an order by. As before, the query enters the system in step 1 and is compiled into query fragments. In step 2, those query fragments are pushed to the nodes that match the distribution function dist(uid<100), in this case, slices T11 and T12. These query fragments will do a container read for all rows matching the constraint uid<100, filter the resulting rows on the constraint gid>10 and stop once it finds 5 rows. The query fragments don't have to do an explicit sort for this query because the data in the slice is already sorted by uid. Those resulting rows are forwarded to node 2 in step 3. On node 2, the result rows are sorted and requested 5 rows are sent back to the session and to the client in steps 4 and 5.

Let's look more closely at what's going on here. As a general rule, the Sierra planner "pushes down" as many calculations as it can. That is to say, it pushes the calculations as close to the data as it can to reduce latency and data transfer. In this example, the filters are placed close to the data and the filtered rows never leave the node. If this were a traditional database with a shared- disk back-end, all those calculations would happen on the initiating node and there would be more data movement, additional locking, and no appreciable parallelism resulting in far less efficient operation. Notice the "limit(5)" appears 3 times, applied close to the data and before sending rows to the client. This is an optimization Sierra will do called "limit push down" to reduce data movement and total calculations required for the query. This same sort of optimization can be applied to other operators like count(*) – count the rows close to the data then aggregate the counts and return.

This query has the same lightweight locking behavior as the previous two examples. The query fragment primitive is general enough that a query of any complexity can be efficiently

 © 2012 Clustrix, Inc. All Right Reserved

Index

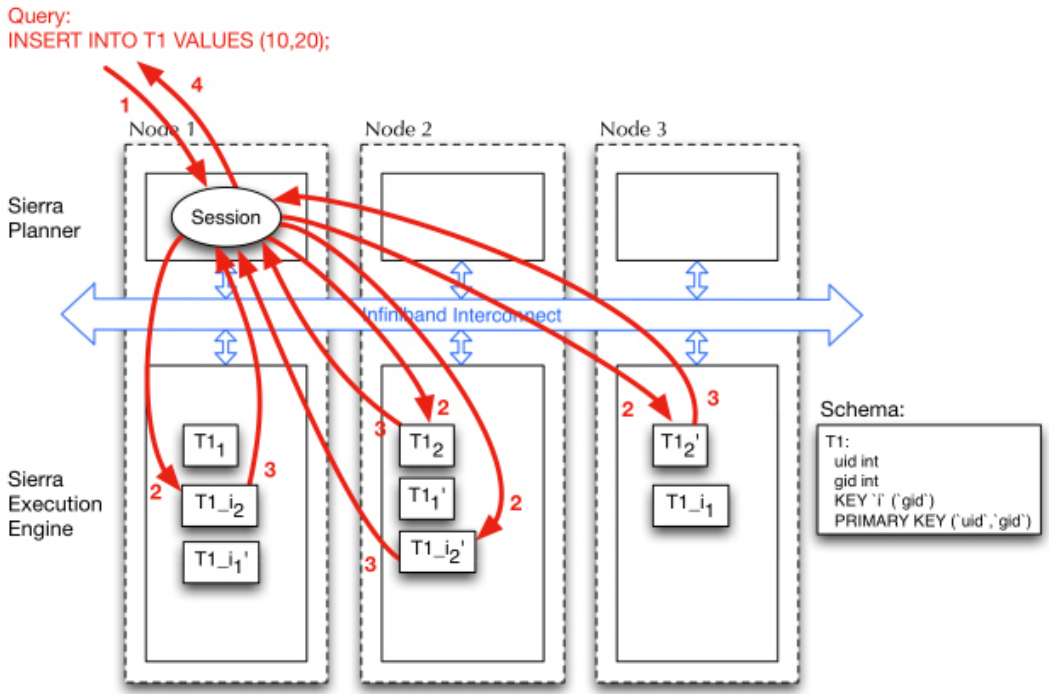

Figure 7: Insert with index

represented to execute with maximum parallelism and concurrency. In fact, as queries become more complicated, Sierra has more opportunity for parallelization and optimization. By moving the query to the data and doing smart optimizations, Sierra enables true scalability and flexibility for simple key/ value lookups as well as complex relational expressions.

Write performance is often the biggest bottleneck of database solutions today. Sierra goes to extreme lengths to provide scalable write as well as read performance. Figure 7 illustrates an insert to a table T1 that contains a secondary index marked as T1_i in the diagram. This diagram includes all replicas of the data since they all get written transactionally. In the Sierra architecture, indices are kept as separate tables internally, each with a distribution based on the columns being indexed. In this example, inserting the row (10,20) needs to update T12 of the base representation and T1_i2 of the index and their replicas. Inserts follow the same flow in Sierra as selects. The query arrives in step 1 and is compiled into query fragments. In step 2, the query fragments and the data are sent to the appropriate slices in parallel. The
and doing smart optimizations, Sierra enables true scalability and flexibility for simple key/ value lookups as well as complex relational expressions. acknowledgements are sent back to the session and client respectively in steps 3 and 4. When the transaction is committed, Sierra uses an optimized parallel distributed commit protocol.

Local locking is used in the insert case as well. Write operations are optimistic. If any of the query fragments sent out in step 2 fail, the transaction machinery will roll back all of the operations. Given Sierra's use of MVCC, no other query will ever see the tables in an inconsistent state. The only locking employed in the insert path is local page-level locking on the on-disk data structure itself, just like in the select case. Just like with the select case, the insert concurrency allowed on the table grows with the table for a truly scalable solution.

© 2012 Clustrix, Inc. All Right Reserved

Conclusion

Move the query to the data, not the data to the query. Clustrix Sierra has taken this simple concept and driven it to the logical conclusion with great benefit. Data movement and global locking are minimized while concurrency and parallelism are maximized. Sierra provides complete flexibility in data layout, which can be changed dynamically without interruption to the user.

Queries ranging from simple point selects to complex joins and aggregates to inserts and updates are fast and efficient using Sierra. Sierra allows the database to scale to any arbitrary size and the performance scales linearly with the data. Clustrix Sierra represents a revolution in database functionality and scalability, while setting-the-bar for Clustered Database Systems.

ABOUT CLUSTRIX, INC.

Clustrix is the scale-out SQL database for Big Data applications. Clustrix provides a radically simple SQL database that enables applications to scale to unlimited users, transactions, and data, while eliminating database sharding and au- tomating fault tolerance. More than 125 Clustrix nodes are in production in WMySQKeb applications around the globe, with more than 500 billion transactions per month running through Clustrix databases worldwide. Customers include Symantec, AOL, MakeMyTrip, Photobox, and Massive Media.

For questions regarding Clustrix, please contact:
info@clustrix.com
Toll Free: 1-877-806-5357
International: +1 415-501-9560