CASE STUDY

Software
Big Data Analytics

intel Software

Streamlining Development for a Real-Time Analytics Database

Intel® Threading Building Blocks (Intel® TBB) enables quasardb to deliver first-class performance for real-time big data analytics

"When we discovered Intel® TBB, we realized it was exactly what we needed. It was written in C++, had the right feature set, was multi-platform, and the performance was very good. Last but not least, with Intel Corporation behind it, we were confident that the library would be supported for a long time and make efficient usage of Intel® architecture, which is the most-used architecture by our customers."

—Edouard Alligand
CEO, quasardb

Quasardb* is a distributed, transactional database designed to give users instant access to data from any kind of analytics engine—enabling companies to do real-time analytics. It can scale both horizontally (across several servers) and vertically (using the capabilities of one server) to deliver first-class performance and outstanding return on investment (ROI). With its scalability, Quasardb gives users virtually unlimited processing capabilities for:

- Efficient and reliable storage of market history
- Real-time market data storage
- Test flight databases
- Logistics delivery history
- And more

Quasardb has been successfully deployed in demanding environments such as finance and aeronautics. "Our customers need to inject terabytes of data into our database in a very short amount of time, usually minutes," explained Edouard Alligand, CEO of quasardb. "In finance especially, they use quasardb to store the whole transaction history. The database has to be able to handle the high throughput while still remaining reliable."

To deliver high performance, quasardb needed to not only do a high-quality implementation, but also to leverage the Intel multi-core infrastructure. One of the key building blocks that enables quasardb to deliver first-class performance is Intel® Threading Building Blocks (Intel® TBB), Intel's C++ threading library for creating high-performance, scalable parallel applications.

Why Multi-Core Matters, Even for I/O-Bound Workloads

Databases are typically I/O bound and benefit from increased memory bandwidth, faster disks, and faster network cards. Quasardb uses fast storage (such as Intel® NVM) and benefits from the large memory bandwidth of the latest Intel® Xeon® processors.

But multi-core architecture is also key to delivering future-proof performance. Through upscaling, users can maximize their server investments, needing fewer servers to handle the same workload. Users can also benefit from upgrading to a processor with more cores.

Upscaling can be achieved in two ways:

1. **Shared nothing architecture** where each core has its own copy of the data to work on. This can deliver linear scalability for certain workloads but may result in greatly increased memory consumptions for some other workloads.
2. **Multi-core aware architectures** that make use of concurrent data structure to minimize—and even prevent—thread stalls.

Quasardb uses the second method.

Simplified Multi-Core-Aware Architecture

Quasardb data is available to all threads; however, each remote request is processed in a dedicated thread. Quasardb is carefully designed to ensure an operation running in one thread does not interfere with another. For example, two reads on the same entry cause no contention. The same is true of two writes to different entries.

This is achieved with the use of Intel TBB concurrent data structures and lightweight locks (Figure 1).

Why Intel TBB?

When the team started designing quasardb in early 2009, it had two choices:

1. **Implement all the fundamental bricks** required for multi-core scalability, either by writing everything themselves or by assembling open source bricks.
2. **Accelerate development** by using a library that provides a comprehensive toolbox to achieve multi core efficiency.

"When we discovered Intel TBB, we realized it was exactly what we needed," said Alligand. "It was written in C++, had the right feature set, was multi-platform, and the performance was very good. Last but not least, with Intel Corporation behind it, we were confident that the library would be supported for a long time and make efficient usage of the Intel® architecture, which is the most-used architecture by our customers."

Building a Multi-Core, Efficient Database

In quasardb, once the engine receives a network request, it's essential to make sure the request is processed within the same thread with as few stalls as possible. Common bottlenecks that can prevent an application from scaling include:

- **Concurrent data structures** to achieve efficient concurrent access to shared data
- **Efficient memory allocation** to prevent bottlenecks caused by an allocator which isn't scalable
- **Lightweight locks** for low latency and avoiding thread switching caused by more heavy-duty locks

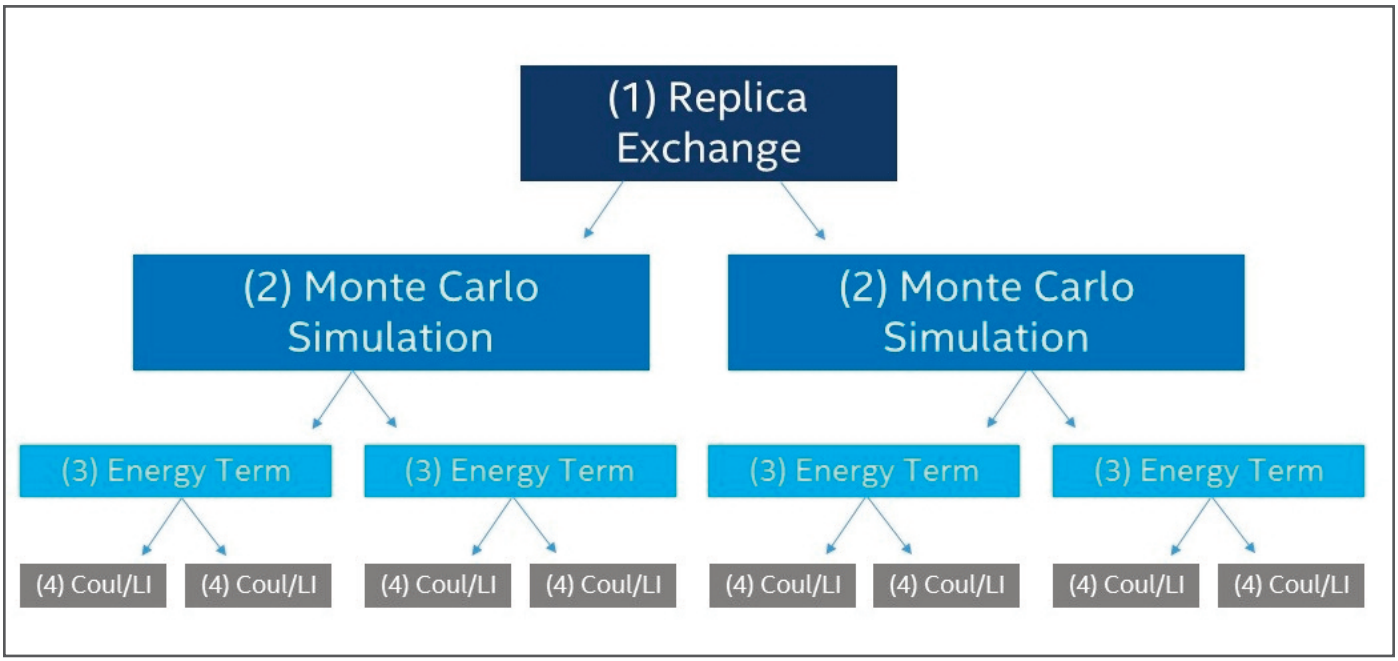

Figure 1. Intel TBB concurrent data structures and lightweight locks

Concurrent Data Structures

Lock-free data structures enable threads to access shared data efficiently. Concurrent data structures reduce contention with simple-to-understand semantics and reduced data duplication. Writing a data structure that is both concurrent and safe can be a long and difficult task.

Intel TBB simplifies that task, coming out of the box with concurrent vectors, queues, sets, and maps. Quasardb makes an intensive usage of those data structures to achieve scalability—for example, in storing elements due for eviction in a concurrent queue, reducing thread contention.

Efficient Memory Allocation

Memory frugality is a key to performance. But there is no way around dynamic memory allocation, especially in a database. When users are adding new entries, this generally results, either sooner or later, in a memory allocation.

"During the development of quasardb, we realized that the default system allocator could cause contention," explained Alligand. "Using the efficient, scalable allocator from Intel TBB would result in up to 100 percent increased performance for update operations. Having the allocator available on all the platforms we support also makes our lives much easier, since it reduces drastic behavioral changes between platforms."

Lightweight Locks

Although the concurrent structures provided by Intel TBB sometimes internally use lightweight locks, this tool has been greatly effective at reducing latency when a thread has to access a shared resource. If using more heavy-duty mutex such as the one provided by the standard library, there is the risk of losing the remaining of the thread quantum, and therefore of a context switch.

"Although it might sound inefficient to use 'busy waiting' locks to access a shared resource," explained Alligand, "when the lock is held for a short amount of time, and when lock contention is rare, it's actually an extremely efficient way to achieve scalability while having a predictable and easy to understand code."

First-Class Performance

Intel TBB has enabled quasardb to deliver first-class performance and take advantage of the latest and greatest Intel technology innovations. The rich toolkit of TBB enabled quasardb to achieve top-of-the-line performance. Using Intel TBB quasardb can achieve sustained rates greater than 100 GBit/s, as validated with Cisco.

"When you're working on a product like quasardb," said Alligand, "the fundamental bricks take a notoriously long time to get right. Intel TBB saved us almost a year of our development time."

Learn more about Intel Threading Building Blocks >