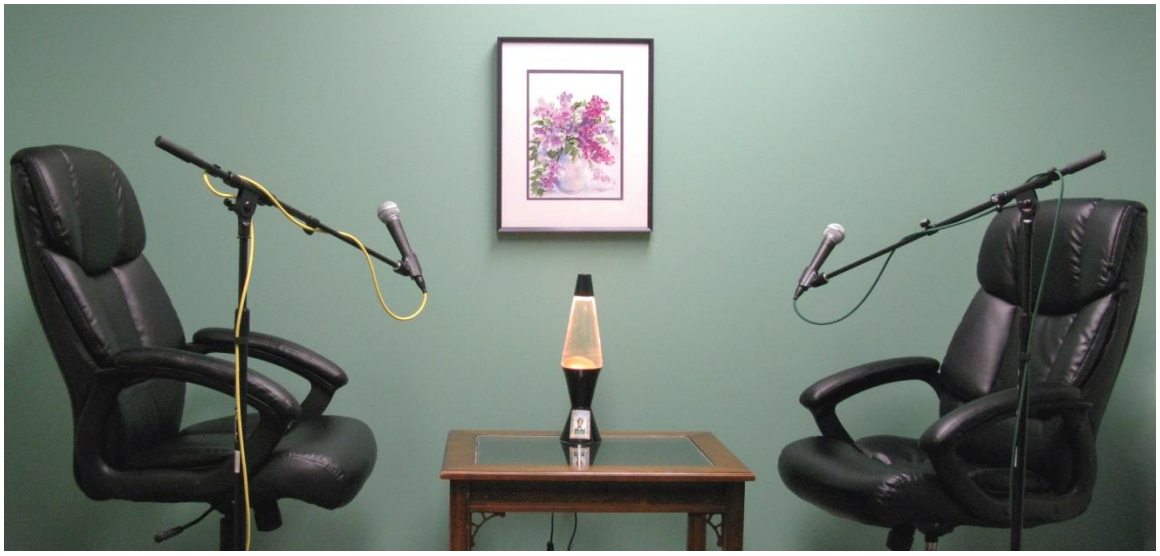
BBBT Podcast Transcript



About the BBBT

The Boulder Business Intelligence Brain Trust, or BBBT, was founded in 2006 by Claudia Imhoff. Its mission is to leverage business intelligence for industry vendors, for its members, who are independent analysts and experts, and for its subscribers, who are practitioners. To accomplish this mission, the BBBT provides a variety of services, centered around vendor presentations.

For more, see: www.bbbt.us.

Vendor:	**NuoDB**
Date recorded:	**July 30, 2014**
Host:	**Claudia Imhoff**, President, BBBT
Guest(s):	**Seth Proctor**, Chief Technology Officer
Run time:	**00:18:17**
Audio link:	Podcast
Transcript:	[See next page]

Claudia Imhoff: Hello, and welcome to this edition of the Boulder BI Brain Trust, or the BBBT. We're a gathering of international consultants, analysts, and experts in business intelligence, who meet with interesting and innovative BI companies here in beautiful Boulder, Colorado. We not only get briefed on the latest news and releases, but we share our ideas with the vendor on where the BI industry is going, and help them with their technological directions and marketing messages. I'm Claudia Imhoff and the BBBT podcasts are produced by my company, Intelligent Solutions.

I'm pleased to introduce my guest today. He Is Seth Proctor. Seth is the Chief Technology Officer for NuoDB. Welcome, Seth.

Seth Proctor: Hey, Claudia. Thanks for having me.

CI: It was a good session today. In fact, let's start off with a little bit about NuoDB. It is a relatively new company. When did it start and who are the major players in the company?

SP: NuoDB was founded by our CEO Barry Morris and our technical founder Jim Starkey. Jim Starkey has been in the database industry just about since there has been a database industry. He's worked in the space for a very long time.

He started thinking about the problem several years ago that we're addressing here at NuoDB. The company started in late 2010. We brought the first version of the product to market in January 2013.

We've had generally available the products about a year-and-a-half now. In terms of the technology resolving, NuoDB is a web-scale distributed database. So, it's a database technology that is designed to solve traditional relational problems around SQL, around ACID, around being able to handle transaction consistency.

It's built around the fundamentally new architecture designed to take advantage of the cloud, being able to do on-demand scale, being able to solve some other interesting problems that people care about when they think about web-scale throughout the cloud issues today.

CI: For those of us perhaps not so knowledgeable in relational DBMS technology, how would you say that your approach is different from the other relational DBMS vendors?

SP: Traditional relational databases really have their lineage going back to the late 1960s and early '70s. They're all designed around a model that works very well from what's called a scale up, or vertical scale.

They all work well for handling more clients or doing more transaction through put by running on larger systems. What they weren't design for was they weren't designed the way the people want the architect systems today. Which is through low-cost, commodity infrastructure, virtualization, being able to scale on-demand, and really being able to run across multiple data centers. So, the difference really is that this is a fundamentally new architecture, because we're architecting our systems end-to-end in a fundamentally new way.

CI: All right. You gave us also a number of customers using NuoDB. They were interesting examples. Maybe you can talk about a couple of them.

SP: Sure, I talked about a couple of customers of ours specifically. They range from very large multi-national, multi-billion dollar companies like Dassault Systems, to companies that are smaller but are growing rapidly like companies like Dropship.

There are many different verticals. They are in communications. They're social networking with games, they're in energy. They're in finance. They're in all of the places where people really care about those cloud properties, but wher they also care about transaction consistency.

Some customers are coming to us, because they have existing applications that work with SQL databases, and want to get them into that scale-out model. Other people are trying to simplify their deployment models and costs, while still insuring they have transactional consistency and they have strong core technology to build against.

It's customers from small start-ups, that are trying to be very agile and still have enterprise features, to enterprise organizations that are really trying to understand how to bring their entire portfolio into modern architectures.

CI: Boy, that's excellent. You pretty much spanned the whole industry, it sounds like.

A couple of other questions, then. Your architecture is a peer to peer one, with the peer's purely in memory. At least that's the way I understand it. If you don't mind talk about your architecture a little bit.

SP: That is exactly why I called it. I mentioned earlier that this is a fundamentally new architecture. What a NuoDB database looks like when it's running is a collection of peers. What I mean by peers is that it's a collection of processes that are all equal.

There's no central coordinator. There's no master keyer. There's no sense of giving process only being responsible for certain set of data. In the current database terminology, we're "active everywhere."

The important difference between the peers is just that some of the peers are running their mode that is purely in memory. Those peers are responsible for handling sequel connections, they're responsible for driving transactions. They're the peers in the system that understand rules of atomicity, consistency, and isolation.

There are other peers that are focused only on data durability. They're responsible for making data durable and providing access to data when it's not available at those memory queues.

You can kind of look at it like a two tier architecture. One tier that's in memory. It's running at memory speeds and is a caching tool. It simplifies your architecture. There is no additional cache you're going to run.

Then there is a second tier that's focused on data durability. You can provision these two tiers independently, scale them independently, you can use them to solve different problems operationally, and also get efficient in terms of resource allocation, and thinking about what you have to buy or what you need to bring online to solve the given database problem.

You mentioned, specifically, technologies like MVCC. Internally we use some technologies that are reasonably well understood, but are critical for making a system like this scale.

MVCC, Multi Version Concurrency Contro,l is a mechanism that says all data in the system has some version. Every time you do an update or every time you do a delete, what you're doing is creating a new record with a new version that's saying, "Here's the change."

Because of the needs of MVCC, we can be very optimistic, we can be highly asynchronous, we are much more efficient and simple at the storage tier, because to think about this, an append-only system. We're never changing anything in place, we're putting changes out as they happen. We can communicate differences in the system very efficiently.

We can really take advantage of this distributed in-memory architecture, what we call the durable distributed cache. We can take advantage of it very efficiently without flooding the network, and we can scale very effectively, in ways hard to do with traditional databases.

CI: All right. "In-memory" brings up a number of qualms in some people's brains. The durability, is that more about high availability? Also, what happens with disaster recovery? What do you do in that situation?

SP: Right, and that's exactly the right set of questions to ask, because you want something that's running at in-memory speeds. You want something that is efficient, and is caching, and is taking advantage of how systems are built today, but you don't want to sacrifice true data durability.

One of the important pieces of our system that we'd let users tune is called the "Commit Protocol." Our Commit Protocol is basically the contract between these two tiers, the contract between that in-memory tier and that data durability tier -- about what guarantees must exist before a given transaction is allowed to connect?

I'll give you an example. Let's say a SQL client is connected to one of the in-memory processes and it makes the change to get something that it has in-memory.

In order to acknowledge back to the SQL client that the transaction has succeeded, all the properties of asset need to be met. In-memory, we're maintaining the rules around atomicity, and consistency, and isolation. We also need to know that data durability has been met, and we let our users decide what that means.

A simple example of that is we can require that the single point of storage must acknowledge one of those durability peers, must acknowledge that it has to change it's written to disk, that the change is truly durable. You can also require that two of those peers has written change, or three of those peers.

If you are not as concerned about data durability, you can go to a simpler protocol. You can do something similar to what's called K-Safety in the NoSQL world.

You can require, simply, that the change has been sensed over reliable channels to all of the storage peers. You don't have to wait for acknowledgement. It's a little bit less safe, but it's also lower latency, and in a distributed system, as long as you have a reasonable number of peers, they'll probably all survive and eventually that change is written.

You can also go to the other extreme, and you can run durability at multiple data centers, and you can require that more than one of those data centers, more than one of those regions acknowledges connect. I think that's the other end of spectrum, because now you're increasing latency but you're also improving data safety.

What we've really done is simplify the system. You don't expose many tuning knobs. We don't make the user configure many things. This is a great example of where we exclusively let the user decide, what is the problem they're trying to solve? Where do they fall in that spectrum, between wanting leading edge, low latency and wanting high degrees of data safety?

We do give our users data durability in the true sense, but, arguably, we give them a much more flexible and tunable definition for what the need when they say "data durability."

CI: All right. Well, let's dive into that a little just bit deeper. What do you consider your breakthrough capabilities? What differentiates you dramatically from any of your competitors?

SP: There are a couple things that really are fundamentally different, and again, these are problems that can be solved by throwing together different pieces of technology. I think the important thing about all of these is that what we're doing is out of the box. We're providing these all in a natural architecture, where you get them all for free without having to work on them.

The key capabilities that we provide… First of all, I've already talked about elastic scale, the ability to steal on demand as you need more transactional throughput, or as you need to run in more places for higher availability, and the ability to scale back when you don't need those resources anymore, when you'd rather allocate them to some other database, some other system, or just drop the resources and stop paying for them.

To be able to do that elastic scale, that on-demand scale, on commodity. You can do high-end systems if you want, but you can also do it on the local data center on cheap systems. You can do it on a laptop. You can do it on VMs. You can do it in public cloud.

Because we provide this kind of on-demand scale, we also provide continuous [indecipherable]. We provide traditional high availability in these cases, in the sense that you can run in multiple places. You can replicate data in multiple places, so you can be resilient in the face of failure.

We also provide management capabilities, you automate the process of any new resources online to pick up the slack when things fail. That means that you really can continue to run even as resources or data centers go down.

It also means that expected failures, things like hardware upgrade or software upgrade, or migration to new resources from the data centers, or even to different clouds, is something you can do real time without taking down the application, without affecting the application module itself.

I mentioned a couple times in there hints towards the third important value proposition, which is around geo-distribution, which is around being able to run in multiple locations, both for high availability, but also to be able to provide low latency, so that if you have users who are doing work on the East coast, users on the West coast, let's say you've got some social apps, and your East coast users tend to be clustered to the East coast, or the West coast users tend to be clustered to the West coast, you get really good performance.

You get low latency experience for all those users. They can travel. They can go East coast to West coast to Europe. Their data will continue to be available to them in a transaction consistent, low latency fashion. Without your application having to understand where you're running things.

A fourth key use case that we care about is run multi-tenancy. Without going into all the details, essentially that peer-to-peer architecture management model I've been talking about enables a powerful use case where we can support along traditional multi-tenant problems.

We can also support a model where you're getting true isolation in terms of process management, resource management, where data is written to disk, and how you're managing independent user's databases truly independently. That's powerful.

The fifth use case, the fifth kind of key capability I think is the most important one in some sense, and that's around simplicity.

It's about something we call no-nobs administration. Basically, trying to make it as simple as possible to work with a system like this that is inherently complicated. That's little things, like making sure we don't expose a lot of configuration knobs, making the system very self-aware. It's things like building natural tools that plug into the existing ecosystem, like tools and monitoring mechanisms and whatever else you're used to using.

It's also around building some really fundamentally new and exciting features, like templates, which essentially are a way of starting a distributed database against a specified SLA, and then walking away and letting us monitor and maintain it. It's a lot of operational intelligence folding back into how to keep a database running as spec.

CI: I love it. I love the term no-knobs administration. You should trademark that. Last question, and we've only got a minute or so left. We also had a good discussion about NuoDB supporting both operational and analytical processing, that mixed workload, if you will, of transaction processing along with being able to handle the analytics side of the house.

If you don't mind, spend a minute or two. We've only got about a minute left. Take about a minute or so and talk about this, if you don't mind.

SP: Yeah, absolutely. I think this is one of those things that is increasingly important to users, is simplifying their employment infrastructure and being able to get analytic data, operational intelligence, on their operational databases.

Because of the way our architecture works, because we're both on MVCC, because we have this independent pure model where its peers can be started on a disjoint different types of hardware with different capabilities, because of some of our load-balancing and management mechanisms, we're able to very easily support a model where you can be running both transactional workloads and analytical or other kinds of intelligence workloads against the same database.

We support being able to do that while we're actually running against physically separate processes so that you manage the network differently, you manage the network differently, you manage cache access patterns differently, but it's done very efficiently because of our peer-to-peer model. It doesn't result in thrashing a particular disk. It doesn't result in affecting overall operational throughput of the system.

It greatly simplifies thing,s because now you don't have to worry about some kind of ETL or some kind of transform out into a separate database to get intelligent information about your operations.

A number of our customers are using that today as their value proposition, as saying they have an operational system where their customers now can get insight, the help desk can get insight and sell that to add-on to your service users.

I think that's a really important use case more and more for our customers, for everyone looking at databases.

CI: Yeah, it certainly is the trend today -- moving in that direction. A very interesting conversation.

Unfortunately, we're about out of time, so that's it for this edition of the BBT Podcast. Again, I'm Claudia Imhoff and it's been a great pleasure to speak with Seth Proctor of NuoDB today. Thanks so much, Seth.

SP: Thank you, Claudia. This was great fun.

CI: I hope you enjoyed today's podcast. You'll find more podcasts from other vendors at our web site www.bbbt.us. If you want to read more about today's session, please search for our hash tag on Twitter. That's #BBBT. And please join me again for another interview. Good bye and good business!