Two Cons against NoSQL. Part I.

by Roberto V. Zicari on October 30, 2012

Two cons against NoSQL data stores read like this:

1. It's very hard to move data out from one NoSQL to some other system, even other NoSQL. There is a very hard lock in when it comes to NoSQL. If you ever have to move to another database, you have basically to re-implement a lot of your applications from scratch.

2. There is no standard way to access a NoSQL data store.
All tools that already exist for SQL has to be recreated to each of the NoSQL databases. This means that it will always be harder to access data in NoSQL than from SQL. For example, how many NoSQL databases can export their data to Excel? (Something every CEO wants to get sooner or later).

These are valid points. I wanted to start a discussion on this.
This post is the first part of a series of feedback I received from various experts, with obviously different point of views.

I plan to publish Part II, with more feedback later on.

You are welcome to contribute to the discussion by leaving a comment if you wish!

RVZ
————

1. It's very hard to move the data out from one NoSQL to some other system, even other NoSQL.

Dwight Merriman (**founder 10gen, maker of MongoDB**): I agree it is still early and I expect some convergence in data models over time. btw I am having conversations with other nosql product groups about standards but it is super early so nothing is imminent.
50% of the nosql products are JSON-based document-oriented databases.
So that is the greatest commonality. Use that and you have some good flexibility and **JSON** is standards-based and widely used in general which is nice. MongoDB, couchdb, riak for example use JSON. (Mongo internally stores "**BSON**".)

So moving data across these would not be hard.

1. If you ever have to move to another database, you have basically to re-implement a lot of your applications from scratch.

Dwight Merriman: Yes. Once again I wouldn't assume that to be the case forever, but it is for the present. Also I think there is a bit of an illusion of portability with relational. There are subtle differences in the SQL, medium differences in the features, and there are giant differences in the stored procedure languages.
I remember at DoubleClick long ago we migrated from SQL Server to Oracle and it was a HUGE project. (We liked SQL server we just wanted to run on a very very large server — i.e. we wanted vertical scaling at that time.)

Also: while porting might be work, given that almost all these products are open

TAGS

Analytics, Basho, Big Data, Cindy Saracco, Clustrix, CSV, document stores, Dwight Merriman, Excel, Hadoop, IBM, InfoSphere BigInsights, John Hugg, JSON, MongoDB, MySQL, New and old Data stores, NoSQL, nosql databases, open source, Oracle, Postgres, relational databases, Riak, SQL, Steve Vinoski, VoltDB

About the Author

Roberto V. Zicari

Prof. Roberto V. Zicari is editor of **ODBMS.ORG** (**www.odbms.org**) , the "most up-to-date collection of free materials on object database technology on the Internet". The portal was created to serve software developers in the open source community or at commercial companies as well as faculty and students at educational and research institutions.
ODBMS.ORG is designed to meet the fast-growing need for resources focusing on Big Data, Analytical Data Platforms, Scalable Cloud platforms, Object databases, Object-relational bindings, NewSQL databases, NoSQL datastores, and new approaches to concurrency control.
Roberto is Full Professor of Database and Information Systems at Frankfurt University and representative of the OMG in Europe. Previously, Roberto served as associate professor at Politecnico di Milano, Italy; Visiting scientist at IBM Almaden Research Center, USA, the University of California at Berkeley, USA; Visiting professor at EPFL in Lausanne, Switzerland, the National University of Mexico City, Mexico and the Copenhagen Business School, Danemark.

Search

Tags

Analytics award awards Big Data Caché cloud stores common persistent model patterns database design patterns db4o db4objects document stores Goethe University Frankfurt Google Google BigTable Hadoop IBM ICOODB impedence mismatch International Conference on Object Databases International Conference on Objects and Databases InterSystems Java Java Object Persistence Lecture Notes MapReduce Michael Blaha MongoDB source, the potential "risks" of lock-in I think drops an order of magnitude — with open source the vendors can't charge too much.

Ironically people are charged a lot to use Oracle, and yet in theory it has the portability properties that folks would want.

I would anticipate SQL-like interfaces for BI tool integration in all the products in the future. However that doesn't mean that is the way one will write apps though. I don't really think that even when present those are ideal for application development productivity.

1. For example, how many noSQL databases can export their data to excel? (Something every CEO wants to get sooner or later).

Dwight Merriman: So with MongoDB what I would do would be to use the mongoexport utility to dump to a **CSV** file and then load that into excel. That is done often by folks today. And when there is nested data that isn't tabular in structure, you can use the new Aggregation Framework to "unwind" it to a more matrix-like format for Excel before exporting.

You'll see more and more tooling for stuff like that over time. Jaspersoft and Pentaho have mongo integration today, but the more the better.

John Hugg (**VoltDB** **Engineering**): Regarding your first point about the issue with moving data out from one NoSQL to some other system, even other NoSQL.
There are a couple of angles to this. First, data movement itself is indeed much easier between systems that share a relational model.
Most SQL relational systems, including VoltDB, will import and export CSV files, usually without much issue. Sometimes you might need to tweak something minor, but it's straightforward both to do and to understand.

Beyond just moving data, moving your application to another system is usually more challenging. As soon as you target a platform with horizontal scalability, an application developer must start thinking about partitioning and parallelism. This is true whether you're moving from Oracle to Oracle RAC/Exadata, or whether you're moving from MySQL to Cassandra. Different target systems make this easier or harder, from both development and operations perspectives, but the core idea is the same. Moving from a scalable system to another scalable system is usually much easier.

Where NoSQL goes a step further than scalability, is the relaxing of consistency and transactions in the database layer. While this simplifies the NoSQL system, it pushes complexity onto the application developer. A naive application port will be less successful, and a thoughtful one will take more time.
The amount of additional complexity largely depends on the application in question. Some apps are more suited to relaxed consistency than others. Other applications are nearly impossible to run without transactions. Most lie somewhere in the middle.

To the point about there being no standard way to access a NoSQL data store. While the tooling around some of the most popular NoSQL systems is improving, there's no escaping that these are largely walled gardens.
The experience gained from using one NoSQL system is only loosely related to another. Furthermore, as you point out, non-traditional data models are often more difficult to export to the tabular data expected by many reporting and processing tools.

By embracing the SQL/Relational model, NewSQL systems like VoltDB can leverage a developer's experience with legacy SQL systems, or other NewSQL systems.
All share a common query language and data model. Most can be queried at a console. Most have familiar import and export functionality.
The vocabulary of transactions, isolation levels, indexes, views and more are all shared understanding. That's especially impressive given the diversity in underlying architecture and target use cases of the many available SQL/Relational systems.

MySQL New and old Data stores NoSQL nosql databases object databases object persistence ODBMS ODBMS.ORG ODBTWG OMG On Innovation open source ORM relational databases SQL standards Versant VoltDB

Archives

November 2012
October 2012
September 2012
August 2012
July 2012
June 2012
May 2012
April 2012
March 2012
February 2012
January 2012
December 2011
November 2011
October 2011
September 2011
August 2011
July 2011
June 2011
May 2011
April 2011
March 2011
February 2011
January 2011
December 2010
November 2010
October 2010
September 2010
August 2010
July 2010
June 2010
May 2010
April 2010
March 2010
February 2010
January 2010
December 2009
November 2009
October 2009
September 2009
August 2009

Finally, SQL/Relational doesn't preclude NoSQL-style development models. Postgres, Clustrix and VoltDB support MongoDB/CouchDB-style JSON Documents in columns. Functionality varies, but these systems can offer features not easily replicated on their NoSQL inspiration, such as JSON sub-key joins or multi-row/key transactions on JSON data

1. It's very hard to move the data out from one NoSQL to some other system, even other NoSQL. There is a very hard lock in when it comes to NoSQL. If you ever have to move to another database, you have basically to re-implement a lot of your applications from scratch.

Steve Vinoski (Architect at Basho): Keep in mind that relational databases are around 40 years old while NoSQL is 3 years old. In terms of the **technology adoption lifecycle**, relational databases are well down toward the right end of the curve, appealing to even the most risk-averse consumer. NoSQL systems, on the other hand, are still riding the left side of the curve, appealing to innovators and the early majority who are willing to take technology risks in order to gain advantage over their competitors.

Different NoSQL systems make very different trade-offs, which means they're not simply interchangeable. So you have to ask yourself: why are you really moving to another database? Perhaps you found that your chosen database was unreliable, or too hard to operate in production, or that your original estimates for read/write rates, query needs, or availability and scale were off such that your chosen database no longer adequately serves your application.
Many of these reasons revolve around not fully understanding your application in the first place, so no matter what you do there's going to be some inconvenience involved in having to refactor it based on how it behaves (or misbehaves) in production, including possibly moving to a new database that better suits the application model and deployment environment.

2. There is no standard way to access a NoSQL data store.
All tools that already exists for SQL has to recreated to each of the NoSQL databases. This means that it will always be harder to access data in NoSQL than from SQL. For example, how many noSQL databases can export their data to Excel? (Something every CEO wants to get sooner or later).

Steve Vinoski: Don't make the mistake of thinking that NoSQL is attempting to displace SQL entirely. If you want data for your Excel spreadsheet, or you want to keep using your existing SQL-oriented tools, you should probably just stay with your relational database. Such databases are very well understood, they're quite reliable, and they'll be helping us solve data problems for a long time to come. Many NoSQL users still use relational systems for the parts of their applications where it makes sense to do so.

NoSQL systems are ultimately about choice. Rather than forcing users to try to fit every data problem into the relational model, NoSQL systems provide other models that may fit the problem better. In my own career, for example, most of my data problems have fit the key-value model, and for that relational systems were overkill, both functionally and operationally. NoSQL systems also provide different tradeoffs in terms of consistency, latency, availability, and support for distributed systems that are extremely important for high-scale applications. The key is to really understand the problem your application is trying to solve, and then understand what different NoSQL systems can provide to help you achieve the solution you're looking for.

1. It's very hard to move the data out from one NoSQL to some other system, even other NoSQL. There is a very hard lock in when it comes to NoSQL. If you ever have to move to another database, you have basically to re-implement a lot of your applications from scratch.

Cindy Saracco (IBM Senior Solutions Architect) (these comments reflect my personal views and not necessarily those of my employer, IBM) :

Meta

Since NoSQL systems are newer to market than relational DBMSs and employ a wider range of data models and interfaces, it's understandable that migrating data and applications from one NoSQL system to another — or from NoSQL to relational — will often involve considerable effort.
However, I've heard more customer interest around NoSQL interoperability than migration. By that, I mean many potential NoSQL users seem more focused on how to integrate that platform into the rest of their enterprise architecture so that applications and users can have access to the data they need regardless of the underlying database used.

2. There is no standard way to access a NoSQL data store.
All tools that already exists for SQL has to recreated to each of the NoSQL databases. This means that it will always be harder to access data in NoSQL than from SQL. For example, how many noSQL databases can export their data to excel? (Something every CEO wants to get sooner or later).

Cindy Saracco: From what I've seen, most organizations gravitate to NoSQL systems when they've concluded that relational DBMSs aren't suitable for a particular application (or set of applications). So it's probably best for those groups to evaluate what tools they need for their NoSQL data stores and determine what's available commercially or via open source to fulfill their needs.
There's no doubt that a wide range of compelling tools are available for relational DBMSs and, by comparison, fewer such tools are available for any given NoSQL system. If there's sufficient market demand, more tools for NoSQL systems will become available over time, as software vendors are always looking for ways to increase their revenues.

As an aside, people sometimes equate **Hadoop**-based offerings with NoSQL.
We're already seeing some "traditional" business intelligence tools (i.e., tools originally designed to support query, reporting, and analysis of relational data) support Hadoop, as well as newer Hadoop-centric analytical tools emerge.
There's also a good deal of interest in connecting Hadoop to existing data warehouses and relational DBMSs, so various technologies are already available to help users in that regard . . . . IBM happens to be one vendor that's invested quite a bit in different types of tools for its Hadoop-based offering (**InfoSphere BigInsights**), including a spreadsheet-style analytical tool for non-programmers that can export data in CSV format (among others), Web-based facilities for administration and monitoring, Eclipse-based application development tools, text analysis facilities, and more. Connectivity to relational DBMSs and data warehouses are also part IBM's offerings. (Anyone who wants to learn more about BigInsights can explore links to articles, videos, and other technical information available through its **public wiki**.)

Related Posts

- **On Eventual Consistency– Interview with Monty Widenius. by Roberto V. Zicari on October 23, 2012**

- **On Eventual Consistency– An interview with Justin Sheehy. by Roberto V. Zicari, August 15, 2012**

- **Hadoop and NoSQL: Interview with J. Chris Anderson by Roberto V. Zicari**

##

From → uncategorized

One Comment Leave one →

Dražen Lučanin permalink

I think the biggest issue with porting data from a NoSQL DB such as MongoDB to a traditional SQL DB such as MySQL is that Mongo is schemaless – i.e. it allows you to create structures which don't fit into a traditional relational DB.

Personally, I don't really see this as a drawback, because if you build your application relying on this "slackness" feature, you obviously did it intentionally by skipping some strict organisation in the beginning. You would have to do it later on before fitting this data in a more firmly structured DB.

Leave a Reply

Name (required)

Email (required, will not be published)

Website

Comment

Note: HTML is allowed. Your email address will not be published.

Subscribe to this comment feed via RSS

Submit Commen

Spam protection by WP Captcha-Free

About

"This is the ODBMS Industry Watch blog. --Trends and Information on New Data Management Technologies, Innovation."

To see the complete ODBMS website with useful articles, downloads and industry information, please CLICK HERE.

October 2012

M	T	W	T	F	S	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

« Sep Nov »

Search

Search