What NoSQL Store Should I Use? The Right Tool for Your Use Case
Submitted by Mitchell Pronsc... on Sun, 2010/07/25 - 11:00pm

Tags:

* Architecture
* Database
* Heroku
* NoSQL
* RDBMS
* Grid Computing

As NoSQL data models continue to prove their worth in high-profile web properties and enterprise settings, developers and architects need a basic framework that helps them organize and differentiate these data stores according to their capabilities so that they can find out where to direct their more in-depth research.

Nathan Hurst's Visual Guide to NoSQL Systems is an excellent way to start your search for one, or many solutions. However, with the emergence of polyglot persistence, you don't necessarily need to choose just one data store, and now that we have data persistence available in hosted services, what's to stop us from trying many systems by popping them in and out of our custom platform? Heroku's Adam Wiggins recently wrote a great post that indicated which persistence solutions might work best in certain use cases:

* Frequently-written, rarely read statistical data (for example, a web hit counter) should use an in-memory key/value store like Redis, or an update-in-place document store like MongoDB.
* Big Data (like weather stats or business analytics) will work best in a freeform, distributed db system like Hadoop.
* Binary assets (such as MP3s and PDFs) find a good home in a datastore that can serve directly to the user's browser, like Amazon S3.
* Transient data (like web sessions, locks, or short-term stats) should be kept in a transient datastore like Memcache. (Traditionally we haven't grouped memcached into the database family, but NoSQL has broadened our thinking on this subject.)
* If you need to be able to replicate your data set to multiple locations (such as syncing a music database between a web app and a mobile device), you'll want the replication features of CouchDB.
* High availability apps, where minimizing downtime is critical, will find great utility in the automatically clustered, redundant setup of datastores like Casandra and Riak.

Nathan Hurst's Visual Guide to NoSQL Systems uses a triangular diagram to visualize the specialization domains of each data model and its tradeoffs.

from Nathan Hurst's Blog

Each corner represents a primary attribute of a system, and every model covers two of the three attributes:

* Consistency means that each client always has the same view of the data.
* Availability means that all clients can always read and write.
* Partition tolerance means that the system works well across physical network partitions.

RDBMSs (Postgres, MySQL, etc.) have both Consistency and Availability
Cassandra, Voldemort, CouchDB, and Riak are some of the models that have Availability and Partition tolerance.
BigTable, HBase, MongoDB, Berkeley DB, and Redis are some of the models that have Consistency and Partition tolerance.

This is a pretty general differentiation of the different data models, but it's definitely a useful tool to direct your research. You'll have to drill down into a lot more research and then do some actual testing of the solution to see if it meets all of your needs. And even if one solution doesn't meet all of your needs, there's also the growing trend of polyglot persistence - give it a try. Another new and growing trend is DaaS - Database-as-a-Service. Basically, as NoSQL shows its natural suitability for cloud computing persistence, more people are able to test them out (at-will) as a cloud-based service.

Heroku's 'drop in and run' cloud platform already supports several add-ons for NoSQL databases. They include: MongoHQ for MongoDB, Cloudant for CouchDB, NorthScale's Memcached service, and soon Redis To Go. Amazon has an RDS service for those who want to stick with MySQL and use it as a hosted service.

More information on choosing the right NoSQL for the job can be found here and here. You can find more info about Heroku and its NoSQL solutions on the referenced article: NoSQL, Heroku, and You.
Your rating:
0

* Login or register to post comments
* 6754 reads
* Printer-friendly version

Comments
Fabrizio Giudici replied on Mon, 2010/07/26 - 3:27am
I wonder why when enumerating NOSQL solutions people don't include RDF stores...

* Login or register to post comments

Peter Veentjer replied on Mon, 2010/07/26 - 4:41am

I have some worries about the whole NoSQL movement, I have the impression that people jump on the bandwagon without realising what they are going to loose.

E.g. failure atomicity... What are the practices/patterns for dealing with partially committed transactions? Reasoning about such a systems becomes orders or magnitudes more complex than when full failure atomicity is available.

And of course consistency: with traditional rdbm's most people don't understand how isolation works, but are protected by isolation levels (and people still get it wrong). But with the NoSQL movement every database does it completely different than the next. This means that a developer needs to have detailed knowledge about a specific database. The question is if developers really do, or just ignore it because they haven't encountered any problems locally (this is the problem I see developers have with traditional databases, so I expect the same with a NoSQL database).

I have no problem with moving away from traditional databases, but I do have problems with people not clearly understanding what they are getting themselves into. It all sounds cool and hot... but your company can burn its fingers too if an application starts to behave unpredictable in production.

Peter Veentjer

Multiverse: Software Transactional Memory for Java

http://multiverse.codehaus.org

* Login or register to post comments

Jakob Jenkov replied on Mon, 2010/07/26 - 5:25am

@Peter

I understand your point, but is that not "just" a period of innovation, gaining experience with what works, and later standardization, that the industry must go through on the move to distributed architectures? NoSQL databases causes problems - yes - and by being exposed to these problems, we learn how to deal with them. Not today or tomorrow, but eventually.

* Login or register to post comments

Fabrizio Giudici replied on Mon, 2010/07/26 - 5:44am
I understand your points, but it's high time somebody took some risks. Not taking risks is the main reason for which we got stuck with SQL for decades... After all, the whole job of a Software Architect is to face with risks. Of course, this must be done in savvy mode and with a good comprehension of what is going on.

* Login or register to post comments

Peter Veentjer replied on Mon, 2010/07/26 - 6:27am
I think a calculate risk can be good.. but even for me... with quite some baggage on board about consistency (the heart of every STM) and ACID related stuff.. a lot of questions remain unanswered. PS: I'm currently also working on a storage solution behind the STM. But it will be a more traditional ACID solution. :)

* Login or register to post comments

Ronald Miura replied on Mon, 2010/07/26 - 8:10am in response to: alarmnummer
@Peter I completely agree.

Most NoSQL articles out there talk about the wonders of NoSQL, even if they are just half-true, but don't say a word about what you had in the relational world, and won't in this new, wonderful place that is the NoSQL world.

You don't have a formal schema, but you have a data format, and changing it requires some kind of migration (which, by the way, will most likely be much more complicated than 'ALTER TABLE items RENAME COLUMN name TO description'). No schema migration? Bullshit.

Well, maybe everybody out there now only create Google- or Facebook-scale apps, while I'm part of the very limited niche of internal enterprise applications which don't need to scale to millions of users, and have to do some reporting and ad-hoc queries.

* Login or register to post comments