101 Questions to Ask When Considering a NoSQL Database

Wednesday, June 15, 2011 at 8:08AM

Todd Hoff in nosql

You need answers, I know, but all I have here are some questions to consider when thinking about which database to use. These are taken from my webinar What Should I Do? Choosing SQL, NoSQL or Both for Scalable Web Applications. It's a companion article to What The Heck Are You Actually Using NoSQL For?

Actually, I don't even know if there are a 101 questions, but there are a lot/ way too many. You might want to use these questions as kind of a NoSQL I Ching, guiding your way through the immense possibility space of options that are in front of you. Nothing is fated, all is interpreted, but it might just trigger a new insight or two along the way.

Where are you starting from?

- A can do anything green field application?
- In the middle of a project and worried about hitting bottlenecks?
- Worried about hitting the scaling wall once you deploy?
- Adding a separate loosely coupled service to an existing system?
- What are your resources? expertise? budget?
- What are your pain points? What's so important that if it fails you will fail? What forces are pushing you?
- What are your priorities? Prioritize them. What is really important to you, what must get done?

- What are your risks? Prioritize them. Is the risk of being unavailable more important than being inconsistent?

What are you trying to accomplish?

- What are you trying to accomplish?
- What's the delivery schedule?
- Do the research to be specific, like Facebook did with their messaging system:

  Facebook chose HBase because they monitored their usage and figured out what was needed: a system that could handle two types of data patterns.

Things to Consider...Your Problem

- Do you need to build a custom system?
- Access patterns: 1) A short set of temporal data that tends to be volatile 2) An ever-growing set of data that rarely gets accessed 3) High write loads 4) High throughput, 5) Sequential, 6) Random
- Requires scalability?
- Is availability more important than consistency, or is it latency, transactions, durability, performance, or ease of use?
- Cloud or colo? Hosted services? Resources like disk space?
- Can you find people who know the stack?
- Tired of the data transformation (ORM) treadmill?
- Store data that can be accessed quickly and is used often?
- Would like a high level interface like PaaS?

Things to Consider...Money

- Cost? With money you have different options than if you don't. You

can probably make the technologies you know best scale.
- Inexpensive scaling?
- Lower operations cost?
- No sysadmins?
- Type of license?
- Support costs?

Things to Consider...Programming

- Flexible datatypes and schemas?
- Support for which language bindings?
- Web support: JSON, REST, HTTP, JSON-RPC
- Built-in stored procedure support? Javascript?
- Platform support: mobile, workstation, cloud
- Transaction support: key-value, distributed, ACID, BASE, eventual consistency, multi-object ACID transactions.
- Datatype support: graph, key-value, row, column, JSON, document, references, relationships, advanced data structures, large BLOBs.
- Prefer the simplicity of transaction model where you can just update and be done with it? In-memory makes it fast enough and big systems can fit on just a few nodes.

Things to Consider...Performance

- Performance metrics: IOPS/sec, reads, writes, streaming?
- Support for your access pattern: random read/write; sequential read/write; large or small or whatever chunk size you use.
- Are you storing frequently updated bits of data?
- High Concurrency vs High Performance?
- Problems that limit the type of work load you care about?
- Peak QPS on highly-concurrent workloads?
- Test your specific scenarios?

Things to Consider...Features

- Spooky scalability at a distance: support across multiple data-centers?
- Ease of installation, configuration, operations, development, deployment, support, manage, upgrade, etc.
- Data Integrity: In DDL, Stored Procedure, or App
- Persistence design: Memtable/SSTable; Apend-only B-tree; B-tree; On-disk linked lists; In-memory replicated; In-memory snapshots; In-memory only; Hash; Pluggable.
- Schema support: none, rigid, optional, mixed
- Storage model: embedded, client/server, distributed, in-memory
- Support for search, secondary indexes, range queries, ad-hoc queries, MapReduce?
- Hitless upgrades?

Things to Consider...More Features

- Tunability of consistency models?
- Tools availability and product maturity?
- Expand rapidly? Develop rapidly? Change rapidly?
- Durability? On power failure?
- Bulk import? Export?
- Hitless upgrades?
- Materialized views for rollups of attributes?
- Built-in web server support?
- Authentication, authorization, validation?
- Continuous write-behind for system sync?
- What is the story for availability, data-loss prevention, backup and restore?
- Automatic load balancing, partitioning, and repartitioning?
- Live addition and removal of machines?

Things to Consider...The Vendor

- Viability of the company?
- Future direction?
- Community and support list quality?
- Support responsiveness?
- How do they handle disasters?
- Quality and quantity of partnerships developed?
- Customer support: enterprise-level SLA, paid support, none

Article originally appeared on High Scalability (http://highscalability.com/).

See website for complete article licensing information.