Should Web Apps "Just Say No" to SQL?

Pros and Cons of Non-SQL Patterns | by Nati Shalom, CTO

The recent inaugural get-together of the NOSQL community indicates a growing anti-SQL sentiment among developers. The NOSQL radicals are not alone: top companies in the Internet world, including Google, Amazon and Facebook, are basing their web applications on alternatives to the traditional SQL database. In this paper I'll briefly review what is driving this trend, survey alternative approaches to SQL databases, and discuss not only their benefits but also the risks and caveats for real-life web applications. Ending the paper is a thought exercize showing how Twitter's widely-publicized scalability problems could be addressed using non-SQL patterns [1].

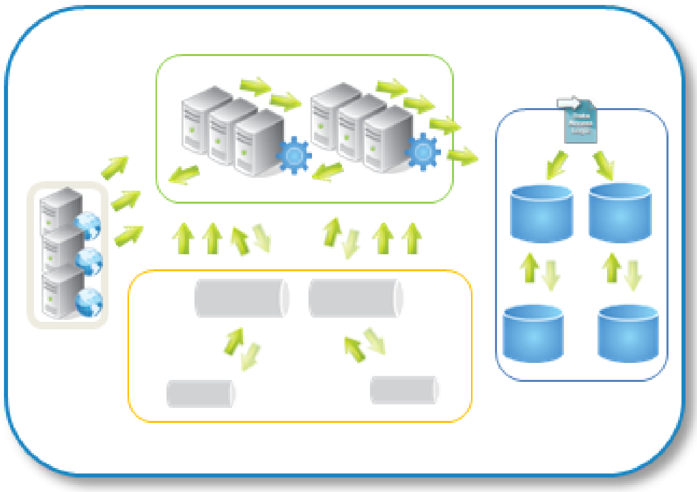
DRIVERS FOR ADOPTING NON-SQL PATTERNS

Two of the biggest assumptions behind the use of databases in applications are being strongly challenged today. The first is the use of a traditional SQL database. The new and growing NOSQL movement claims that **simple key/value stores running on commodity hardware** and designed for massive scale, are the modern replacement for the database. One of the important principles of the NOSQL movement is that **failure is inevitable**, and we should build systems that reduce the impact of failure [2].

The second assumption is that disk is a reliable form of storage. Many organizations take it for granted that their data is "safe" if it is stored on a disk. But according to empirical research, physical disks are not as reliable as many of us think. Actual disk failure/year as measured in the field is up to **600% higher than failure estimates** provided by vendors (3% actual failure vs. 0.5-0.9% estimated). Amazingly, there is no correlation between type of disk (SCSI, SATA, fiber channel) and failure rates – high-end hardware is not more reliable. And lower disk temperatures are actually associated with ***higher*** failure rates [3].

A recent Computerworld article entitled "No to SQL? Anti-database movement gains steam" (Eric Lai, July 2009) points to three main drivers that lead companies like Google, Amazon and Facebook to choose an alternative approach:

- **Demand for extremely large scale** – it is well known that the centralized database quickly becomes a bottleneck when data-intensive applications scale up. This is because the data is stored on a physical disk, which has inherently high latency, and because databases cannot be scaled up easily and can reach very high levels of contention during peak loads.
- **Complexity and cost of setting up database clusters** – "sharding" a database, which involves cutting it up into multiple tables to run on large clusters or grids, is a complex process that requires hard work and special skills. Non-SQL solutions can run on regular PC clusters, which are cheap and easy to expand.


Traditional, centralized software architecture

- **Better performance, even at the cost of reliability** – a good example is HTTP Session data, which needs to be shared between several web servers. Because the data is transient (it is no longer needed when the user logs off) there is no need to store it in the database. However, for many applications this can cause inconsistency and make it very difficult to recover, in case of a widespread failure which interrupts customer transactions.

The classic early adopters of NOSQL patterns are those who hit the scalability wall when their applications simply cannot scale further without unbearable cost. It is very likely that as these alternative solutions mature, they will find their way into mainstream development as well. The SQL database is not going away. but awareness will grow of usage scenarios in which it has failed, and needs to be replaced by the emerging alternatives.

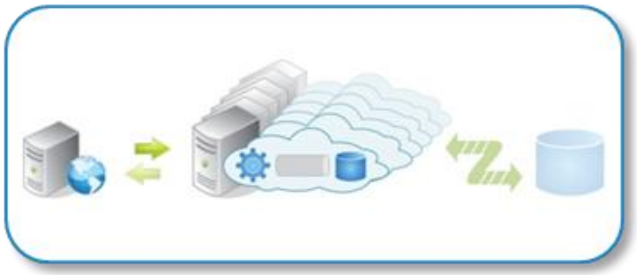
WHAT ARE THE ALTERNATIVES?

The solutions most frequently mentioned in the context of non-SQL patterns are Dynamo, Cassandra, Voldemort, Amazon's SimpleDB, Google's App Engine Megastore, CouchDB, and Microsoft's Azure Table. But the reality is that none of them can or should be positioned as a direct alternative to your existing database – a good case in point is SimpleDB [4]. The main reason is that all of them offer weak or eventual consistency, which is unacceptable for many if not most database-driven applications.

Another category of alternative solutions are the in-memory data grids, such as memcached, GigaSpaces XAP, Oracle Coherence, IBM eXtreme Scale, etc. These solutions front-end the database with an in-memory cluster, which becomes the application's "system of record" and uses the SQL database as the background persistent store.

The main advantage of this model is that it *is* comparable to a traditional database – all the commercial data grid products (excluding memcached) guarantee more or less the same level of consistency and reliability as a traditional database.

Among the data grid solutions, GigaSpaces XAP is unique in that it offers an even higher level of reliability than traditional databases. This is because it partitions and scales not only the data, but also the applications themselves and all the middleware they require to function, thus guaranteeing uptime for the entire application [5].


Space-Based Architecture

XAP does this using Space-Based Architecture (SBA), a software architecture pattern for stateful, high-performance applications, inspired by Yale university's Tuple Space model. Using SBA, XAP runs the entire application, from the load balancer on the front-end to the database on the back-end, in a lightweight scalable container, running entirely in-memory. Maintaining a backup of this container on another machine guarantees uptime for the entire application stack. By adding more containers, it is possible to linearly scale the application, with very low latency at any scale, because all middleware components are colocated.

Alternative Solutions Roundup – Change Required, Value, Storage Cost

When considering a data scalability solution (whether SQL-based or NOSQL), you must be aware of three important factors: the change required to existing applications, the value delivered by the solution at the end of the day, and the cost of storage implied by the solution.

Database clustering solutions are thought to be easy to implement, relatively to NOSQL solutions, because they leverage existing databases. But in reality, these solutions split up the database into a master copy and several read-only replicas, and applications need to be modified to write to the master and read from one of the replicas. This change needs to happen across the board, because the database is shared by many applications, including back-end, analytics, etc.

In terms of value, database clustering only affects scalability of read operations: it does not affect scalability of write operations and does not improve performance or latency (compared to a single database). Database clustering also imposes a high storage cost, because it typically relies on expensive high-end servers.

As for disk-based NOSQL solutions, it is well known that BigTable, Dynamo, Cassandra and the like require a complete rewrite of traditional applications (to put it differently, they are primarily suitable for green-field applications). On the other hand, these solutions deliver high scalability for both read and write operations, and have the lowest storage cost, because they use the disk as their storage medium and run on inexpensive commodity hardware.

The in-memory NOSQL options, because of their built-in support for traditional databses, are much easier to plug into existing applications. They require a moderate integration effort, but this is limited to a single application – back-end apps can continue using the database as usual, and are not affected by the NOSQL solution. This means that implementing in-memory NOSQL solutions can actually be *easier* than implementing a database cluster.

In-memory NOSQL solutions deliver the highest value – they provide unlimited read and write scalability, and the highest read/write performance, because data is accessed at in-memory speeds. The downside is the high cost of memory storage. But according to a Standford University study:

> “RAM clouds [in-memory data grids –N.S.] become much more attractive for applications with high throughput requirements. When measured in terms of cost per operation or energy per operation, RAM clouds are 100-1000x **more** efficient than disk-based systems” [6]

This is an important point – storage cost is not absolute. If you can perform many more operations on 100GB of memory as opposed to 100GB of disk space, that 100GB of memory gives you far more value for money (you could get more work done, or buy less hardware) – by a factor of up to 1000x.

The following table summarizes the change required, value and cost of each class of solution.

	Sample Solutions	**Change Required**	**Scalability**	**Performance & Latency**	**Cost**
SQL	**MySQL database clustering**	Moderate, across the board (*)	Only affects read scalability	No change	Moderate storage cost
NOSQL (disk-based)	**BigTable, Dynamo, Cassandra, etc.**	Complete re-write	Read/write scalability	Moderately improves read/write	Low storage cost
NOSQL (in-memory)	**Memcached**	Moderate, application only	Only affects read scalability	Improves read performance only	High storage cost
	GigaSpaces XAP Data Grid, Oracle Coherence, etc.	Moderate, application only	Read/write scalability	Highest read/write performance	High storage cost

CONCERNS WITH ADOPTING NOSQL SOLUTIONS

All the alternative solutions mentioned above are partitioned and distributed by nature, and therefore less complex and much more scalable than the old SQL database. The benefits are obvious – with these alternative solutions, most users can avoid or at least defer hitting the scalability wall.

However, it is highly recommended to take cautious steps toward these alternative solutions. The fact that you can switch between centralized and partitioned topologies without changing your code doesn't mean that your application will behave correctly and will scale as you expect. In an Architect Summit held last summer, it was almost universally agreed that in distributed systems, "The spectrum of failures within a network is entirely different … The application needs to be made aware of latency, distributed failures, etc. … The fact that the system is distributed leaks through the abstraction." (John Davies, 2008).

GigaSpaces recommends designing your data model to fit into a partitioned environment – even if at first you will run on only one server. This will allow you to scale when required, without massive changes.

Another point of concern is that the traditional 2PC (two phase commit) model doesn't fit with many of the distributed data alternatives. Pat Helland of Amazon.com has suggested an alternative distributed transactions model – break the operation into small individual steps where each step can fail or succeed individually [7]. This makes it is easier to ensure that each step can be resolved within a single partition, thus avoiding the overhead of coordination across multiple partitions. This has been one of the core concepts in designing scalable applications with Space Based Architecture (for more details see the Wikipedia entry for *Space-Based Architecture*). It is also the concept behind the Actor model introduced with new functional languages like Scala and Erlang – this concept is built into Space Based Architecture, as implemented by GigaSpaces XAP [8].

STAYING FRIENDS WITH THE SQL WORLD

SQL databases are not going away, but there is definitely a place for a more specialized data management solutions alongside traditional SQL databases. The adoption of these new solutions should be determined by two main factors: **(a) How well they integrate with the SQL world**, and **(b) How easy it will be to develop** for these new alternatives, that is to say, how smooth the transition will be. For example, the in-memory data grid solutions are pre-integrated with SQL databases and enable various levels of synchronization with existing databases. This is extremely important because it minimizes the impact of Non-SQL patterns on existing applications.

A THOUGHT EXERCIZE: SCALING TWITTER USING NON-SQL PATTERNS

Scaling a real-time web application such as Twitter introduces unique challenges that are are quite different from those of a "classic" database-centric application. The most profound difference is that Twitter is a heavy read/write application, and not read-mostly. This seemingly minor difference can break most existing models for web application scalability. Indeed, Twitter has suffered from widely-publicized scalability problems.

Here is how Twitter's scalability issues could be addressed using non-SQL patterns:

- **Write-operations scalability – partition the tweets.** A natural way to partition the tweets is by user ID. This enables spreading the load of different users across partitions. Retrieving the tweets of a certain user will be resolved in one call to a single partition. This enables linear scalability as the number of users grows.
- **Read-operations scalability – blackboard pattern.** Each follower who follows a group of people is polling for recent messages posted by those people. In SQL syntax: *SELECT * FROM Post WHERE UserID=<id> AND PostedOn > <from date>*. Using Map/Reduce, the application executes one call that looks for all the users being followed, and checks for new tweets.
- **Hold recent tweets in memory**, improving performance and avoiding a database bottleneck. Older tweets can be dumped to the SQL database in the background and accessed only when needed.

>> For more details, see our white paper, Designing a Scalable Twitter with Space-Based Architecture (available at www.gigaspaces.com/whitepapers)

REFERENCES AND FURTHER READING

[1] Nati Shalom's blog (http://natishalom.typepad.com), *No to SQL? Anti-database movement gains steam – My Take*.

[2] For more on the relevance of NOSQL solutions for mainstream applications, see Nati Shalom's blog, *The Common Principles Behind NOSQL Alternatives*.

[3] For a review of this literature and its implications, see Nati Shalom's blog (natishalom.typepad.com), *Why Existing Databases (RAC) are So Breakable*.

[4] Nati Shalom's blog, *Amazon SimpleDB is Not a Database*.

[5] For more details, see *GigaSpaces.com*, "GigaSpaces XAP In-Memory Data Grid Solution", www.gigaspaces.com/datagrid and "Zero Unplanned Downtime Under Unpredictable Load", www.gigaspaces.com/availability

[6] Ousterhout et. al., *The Case for RAMClouds: Scalable High Performance Storage Entirely in DRAM*, p. 12, emphasis in original.

[7] See Nati Shalom's blog, *Lessons from Pat Helland: Life Beyond Distributed Transactions*.

[8] See Shay Bannon's blog (www.kimchy.org), *Actor Model and Data Grids*.

U.S. Headquarters
GigaSpaces Technologies Inc.
317 Madison Ave, Suite 823
New York, NY 10017
Tel: 646-421-2830
Fax: 646-421-2859

California Office
GigaSpaces Technologies Inc.
101 Metro Drive, Suite 350
San Jose, CA 95110
Tel: +1-408-878-6982
Fax: +1-408-878-6149

International Office
GigaSpaces Technologies Ltd.
4 Maskit St., P.O. Box 4063
Herzliya 46140, Israel
Tel: +972-9-952-6751
Fax: +972-9-956-4410

Europe Office
GigaSpaces Technologies Ltd.
2 Sheraton St.
London, W1F 8BH, United Kingdom
Tel: +44-207-117-0213
Fax: +44-870-383-5135

Asia Pacific Office
GigaSpaces Technologies Ltd.
Level 21, Centennial Tower
3 Temasek Avenue
Singapore 039190
Tel : +65-6549-7220
Fax: +65-6549-7001

Corporate Site: www.gigaspaces.com
Team's Blog: blog.gigaspaces.com
Community Site: www.openspaces.org