NoSQL
EVALUATOR'S GUIDE

McKnight | Consulting Group

William McKnight is the former IT VP of a Fortune 50 company and the author of "Information Management: Strategies for Gaining a Competitive Advantage with Data". He is a columnist at Information Management Magazine, a frequent speaker at the Data Warehousing Institute, a member of the Program Advisory Committee for NoSQL Now! and an analyst for GigaOM Research.

TABLE OF CONTENTS

INTRODUCTION

Organizations are increasingly conceding the fact that the exploitation of its big data is a major factor in competitiveness in the next decade. Getting its big data under management is step one and, while the relational database market is at an all-time high, NoSQL databases are seen by many as the more elegant way of managing big, and occasionally small, organizational data.

While the temptation always exists to solve today's data issues with yesterday's solutions, most organizations have begun to cross the chasm to NoSQL databases. They are finding a remarkably mature market for its age, with a wealth of products to choose from.

Many workloads today need to manage all of their sensor data, full webclick data, product data, state data, social data and/or related offshoots for up to thousands of concurrent users needing their data in real-time. This data may not fit well into non-contrived relational database tables. It is these workloads that represent competitiveness for the modern organization - a pulsating system of vibrant activity made possible with NoSQL.

NoSQL is best suited to handle the requirements of these high-performance, web-scalable, big data workloads. There are many examples of companies that use NoSQL databases to gain greater scale and performance, and they do it at less than the cost of relational database systems.

This paper is for technology decision makers confronting the daunting process of selecting from this fast-growing category of data management technologies. It will introduce a set of comparative features that should be used when selecting a NoSQL technology for your workload and your enterprise. There are many common features as well across NoSQL databases, but even these have implementation nuances that should be understood.

COMMON FEATURES OF NoSQL DATABASES

NoSQL databases are generally open source with more scale-out than scale-up capabilities. They are built on clusters made of commodity-class hardware and utilize replication as the primary means of failover. These are worthwhile features, but they also necessitate a specific look at when it comes time for selection. These are the features that are generally found in NoSQL databases and form minor evaluation criteria for selection.

Open Source

The most popular NoSQL databases are lead by innovative, open source communities. However, these databases are also available in enterprise editions with professional support from companies dedicated to the product.

Schemaless

NoSQL solutions do not require, or accept, a pre-planned data model whereby every record has the same fields and each field of a table has to be accounted for in each record. They support a flexible data model. Though there can be strong similarities from record to record, there is no “carry-over” from one record to the next and each field is encoded with JavaScript Object Notation (JSON) or Extensible Markup Language (XML) – according to the solution’s architecture. The result is that developers have the agility they need to meet evolving business requirements. It is this flexible data model that enables these databases to run on multiple servers.

Scalability with Scale Out

NoSQL solutions support a scale out model for growth by dividing the programming across a single data set spread over many machines. While relational databases are engineered to scale up by adding additional resources to the server, NoSQL databases are engineered to scale by adding additional servers or nodes. There is no limit to the amount of servers that NoSQL databases can run on.

Distribution with Sharding

NoSQL databases are engineered to run on multiple installations. NoSQL solutions utilize a partitioning pattern known as sharding that places each partition in potentially separate servers that are potentially physically disparate. The result is that each server is responsible for operating its data instead of all of the data.

Eventual Consistency

Some NoSQL solutions still do not have strong consistency like a single machine system does. Each record will be consistent, but transactions are usually guaranteed to be "eventually consistent" which means changes to data could be staggered for a short period of time due to a lower latency in the write operation.

Commodity Class Nodes

The scale-out NoSQL architectures are built to utilize a "commodity", unspecialized class of computer. These affordable servers comprise a single highly available system, with its servers working in parallel. NoSQL is designed to minimize the cost per measure of performance.

Parallel Query with MapReduce

NoSQL solutions use MapReduce to take a large problem and divide it into locally-processed sub-problems. While relational databases are engineered to execute processes on a single server, NoSQL databases are engineered to execute processes on multiple servers. As a result, these databases can execute processes faster because they can leverage the resources available on multiple servers.

Cloud Readiness

Most NoSQL solutions have a defined cloud computing model since the volume and unpredictability of the data selected for NoSQL make the cloud an ideal place to store the data. The cloud enables NoSQL databases to scale out on demand in an efficient manner.

High Availability

NoSQL databases should support high availability. They should tolerate the failure of any component - a single instance, a physical server, a rack (multiple physical servers), and a data center (multiple racks).

NoSQL DATABASE DATA MODELS

There are some dominant “data models” for NoSQL solutions. These are technical differences in how the tools are implemented but increasingly they do not necessarily constrain the allocation of a NoSQL workload. Nonetheless, it is important to understand the differences.

Distributed Caches

A Distributed Cache may or may not be classified as a NoSQL database. However, it is a critical component of the data tier, implemented alongside a NoSQL database. In order to enable read operations that meet the highest of performance requirements, a Distributed Cache stores the data in memory. When a database becomes the source of performance issues, a Distributed Cache may be required.

Key-Value Stores

A Key-Value Store is a distributed cache that writes the data to disk. The data could be an object, a file, or a document.

Key-Value Stores have no understanding of the value part of the record. It is considered simply a “blob”, keyed by a unique identifier which serves a “primary key” function. This key is used for all get, put and delete. You can also search inside the value, although the performance may be suboptimal.

The commonality in great Key-Value Store uses is that the primary use of the data is the need to look up the full session information at once and to do so by a single key.

The primary purpose of a Key-Value Store is to enable read and write operations that meet very high performance requirements. Document Stores and Wide Column Stores are also “key-value” in the sense of how data is stored.

Document Stores

Document stores are specialized key-value stores. The data is a document formatted in open standard JSON, a human-readable, text-based format for exchanging and consuming data.

Documents (records) can be nested and document stores add the ability to store lists, arrays and sets. It is an excellent model when fields have a variable number of values (i.e., addresses, “likes”, phone numbers, cart items) according to the record. Unlike Key-Value Stores, the value part of the field can be queried. The entire document/record does not need to be retrieved by the key which is excellent when more granular data is required.

The primary purpose of a document database is to enable read AND write operations that meet high performance requirement without the need for an identifier, or key, in order to access the data.

Wide Column Stores

A Wide Column Store is a nested Key-Value Store where each column’s values (or set grouping of columns) are stored independently. The data can be stored in columns, super columns with multiple, similar columns or column families with multiple, related columns.

In column stores, defined column families must exist in each record. This makes the column store ideal for semi-structured data where there is some commonality, as well as differences, record to record. Column families would be comprised of columns that have similar access characteristics.

Column stores are ideal for application state information across a side variety of users. The primary purpose of a column store is to enable read and write operations that meet high performance requirements.

Multi-Layer Database

A Multi-Layer Database layers multiple NoSQL databases into a single solution. The inner layer is a Distributed Cache. The middle layer is a Key-Value Store. The outer layer is a Document Store, a Wide Column store, or any other type of NoSQL database. With this layer consolidation, separate Distributed Caches and multiple NoSQL databases may not be required.

Graph Databases

When relationships are the important aspect to the data, graph databases shine. The data does not have to be “big” for the graph database to provide significant performance benefits over other database technologies. The data itself can be homogenous, such as all people and their relationships as in a “social graph” or heterogenous.

Fast checks of how many degrees nodes are from each other or list pulls of all those a certain number of degrees apart are workloads that would be slow, nested table self-joins in relational databases and probably worse in the other NoSQL data models. Graph Databases yield very consistent execution times that are not dependent on the number of nodes in the graph.

It might be tempting to select from these solutions without giving much “enterprise” thought to the matter. However, an organization that sees the value in a NoSQL database for one application could soon need or use several NoSQL implementations. You could adopt multiple NoSQL databases to satisfy different requirements (e.g. a Document Database and a Key-Value Store). Or you can use a NoSQL database that functions as both a competent Key-Value Store and a competent Document Store. It may also be advantageous to have skills around a single multi-purpose NoSQL database.

EVALUATION CRITERIA FOR A NoSQL DATABASE

The workload match with a NoSQL data model is important in narrowing the field. However, it is perfectly reasonable for that field to still span multiple data models . You may be deciding between a key-value store and a document store for example. Regardless, additional criteria must be applied to make a product selection.

The goal is to deliver ROI to the business with the workload. Ultimately workload success must be delivered so it must be understand how that is measured and look at NoSQL solutions through that lens.

The Ability to Perform

It is primarily the performance of its reads and writes that constitutes the success of a workload. Success by this measure is given a huge step forward with correct product selection. For NoSQL solutions, several types of performance should be tested. These include:

- Throughput latency for all operations
- Read-intensive workloads
- Write-intensive workloads
- Balanced workloads

Significant factors involved in a NoSQL selection for performance include:

Ability to Scale Up

The cluster should have the ability to take on progressive node specifications across the cluster that include modern CPUs, more and improved memory (see below) and more and improved storage. Data nodes should be able to have different specifications within the cluster, either to facilitate a staggered node upgrade process, to accommodate data with different relevancy or to improve the cluster in a budget-conscious way.

Concurrency Control (Locks, Multiversion Concurrency Control, Coarse Grained versus Fine Grained)

NoSQL solutions should provide a rich set of possibilities for locking granularity. Locking should be possible but access to snapshots while data is being written, providing consistent views at all times regardless of concurrent write activity, should be allowed. Locking at all levels, such as document, should be supported and applied according to context.

Object Caching (Integrated versus External)

With a cache, cached data will match the data stored in the database at all times. This cached data can be kept in system memory which is transparent, compared to having an external caching tier. Integrated caching simplifies the architecture by removing a separate failure point.

Ability to Use Memory Efficiently

Engineering performance today means selectively, but productively and generously, using in-memory – the fastest performing storage medium generally available today. In NoSQL solutions, this means utilizing memory efficiently by using the memory available in the cluster to centrally cache and administer data in the address space. Applications can request for memory to be cached and then read it directly off the node's address space for extremely efficient scans by avoiding disk altogether.

Support for Sharding (Reads and Writes)

Sharding is a partitioning pattern that places each partition in potentially separate servers – potentially all over the world. This scale out works well for supporting people all over the world accessing different parts of the dataset with performance. A shard is managed by a single server. Based on the replication schema, the shard is most likely replicated on other shards. Shards can be set up to split automatically when they get too large or they can be more directed. Auto-sharding takes a load off of the programming, which would otherwise not only have to manage the placement of data, but also the application code's data retrieval.

THE ABILITY TO BE AGILE TO THE NEEDS OF THE ORGANIZATION

The workload needs to be up and running quickly. Getting to sustainable production quickly is the second major success category for a NoSQL workload. To be agile, the product must also be agile. Agility is achieved by:

Large Established Community

The product community will largely drive the changes, ensuring the product is agile to the needs of customers. Without a large, active community, product direction will suffer and enhancement will lag. A NoSQL product community extends to the software and platform community that it is enabled for. Having more choices clearly makes the NoSQL product more useful to an organization.

Ease of Installation and Configuration (hours versus days)

Agility with NoSQL requires going from zero to hero quickly, and repeating the feat to higher levels. It requires the ability to install open source software, do some development, move to an enterprise version while keeping that development, and develop some more, providing increasing levels of value. Complex installations and architectural improvements requiring documentation or live support accompaniment are not agile.

Support for Integration with Apache Hadoop

A litmus test for agility is integration with Apache Hadoop, the undisputed preferred analytical big data platform, and what major distributions are based on. While NoSQL manages operational big data, deeper, batch analytics is often on Apache Hadoop and requires integration with the chosen NoSQL platform.

Support for Deployment to Containers, Virtual Machines, and the Cloud

With most organizations moving to the benefits of server virtualization, the NoSQL solution should be tested and able to be deployed to the virtualization platforms of a virtual machine, container and cloud.

Continuous Availability with no single point of failure

Architectural protection against downtime is a must. Dealing with downtime and its recovery after the inevitable component failures are activities outside an agile, streamlined path of continuous improvement. The selected NoSQL system must provide protection against downtime with tested, continuous availability and this must be supported with an architecture without a single point of failure.

Read and Write Anywhere

Location-independent read and write operations allow a NoSQL database to spread data across multiple data centers including clouds. With a peer-to-peer architecture, a NoSQL database provides for both read and write anywhere capability. You can run an application between two data centers to ensure read and write no matter where the usage is located.

THE ABILITY TO SCALE

Finally, if the good performance goes away when the application needs to scale, all would be for naught. The third broad measure of workload success is scale.

The solution should be scalable in both performance capacity and incremental resource growth. The solution should scale in a near-linear fashion and allow for growth in data size, the number of concurrent users, and the complexity of data access.

For NoSQL solutions, this includes all operations dealing with nodes (additions, removals, restarts, temporary disconnects, failures, data center expansion, multi-data center usage, etc.). Understanding hardware, software and procedural requirements for such growth is paramount.

The architecture of a database defines its ability to scale.

NoSQL technology must scale with exponentially increasing data volumes and user requirements. The factors in NoSQL scale include:

Topology (One Node Type versus Multiple Node Types)

While some databases require a simple topology where every node is of the same type, others require a more complex topology where there are multiple types of nodes. A simple topology will be easier to scale since nodes can be added on demand. A complex topology with varied node types and multiple components per node will require more expertise and precision for the scaling process.

Node Addition Process (Disruption, Ease)

Adding nodes requires planning, but ideally it is non-disruptive to applications and as easy to do as assigning an IP address to the node in the range of the cluster. When those nodes are added, the NoSQL engine should adjust any hash function to begin to assign records to the new node in a balanced fashion. The current data should also be rebalanced.

Rebalancing (Controllable versus Not)

The rebalancing process should be automated by the database and it should be fast and uneventful, requiring no change to the applications and no downtime. Ideally the NoSQL database intelligently allows the customer to choose the correct time to rebalance,which may require coordination across data centers.

Administration Console (GUI, Integrated versus Separate)

It is important that administrators can efficiently and effectively manage and monitor the database. This requires both a graphical user interface (GUI) and a command line interface (CLI) that enables administrators to configure and maintain the database. It addition, it should enable administrators to monitor the database via multiple relevant metrics. While some databases require separate management and monitoring software to be installed, others embed it within the database, which enables administrators to manage and monitor the database from any instance.

Mirroring Process (Synchronous versus Asynchronous)

Replication ensures that mirror copies are always in sync. In synchronous mode, after the secondary server responds, the primary server sends an acknowledgement to the client for placing the data in both nodes simultaneously. In asynchronous mode, the updates happen serially, exposing some milliseconds to inconsistent data. Asynchronous mode is “eventually consistent” if the client is allowed to read from the secondary server.

SUMMARY

Whether you need to scale beyond the capabilities of a relational database infrastructure, scale down a budget for the management of information, or both, there are several ways NoSQL databases make sense.

Your NoSQL database choice should depend on your workloads – both known and anticipated. The selection should obviously be immediately useful but should also consider the enterprise.

It's important to understand the differences in the solutions in this category. As discussed in this paper, the details of the common features of NoSQL databases and the data model are determinants of optimal selection. The ability to achieve high performance, get there quickly out of the box and stay there through growth periods are paramount to successful selection.

COMPARISON MATRIX FOR 3 NoSQL DATABASES

	Couchbase Server 2.5.1 Enterprise Edition	Datastax Enterprise 2.0 (Cassandra)	MongoDB Enterprise 2.6.0
Ability to Scale Up	✓	✓	✓
Robust Concurrency Control	✓	✓	
Integrated Object Caching	✓		
Ability to Use Memory Efficiently	✓	✓	✓
Support for Sharding	✓	✓	✓
Large, established community		✓	✓
Easy Installation and Configuration	✓	✓	
Integration with Apache Hadoop	✓	✓	✓
Support for Deployment to Containers, Virtual Machines, and the Cloud	✓	✓	✓
Continuous Availability with no single point of failure	✓	✓	
Read and Write Anywhere Across Data Centers	✓	✓	
One Node Type	✓	✓	
Easy and Non-disruptive node additional process	✓	✓	✓
Controllable Rebalancing	✓	✓	✓
Integrated, GUI Administration Console	✓	✓	✓
Synchronous Mirroring	✓	✓	✓