and having descriptive key names actually hurts since they increase the document size.

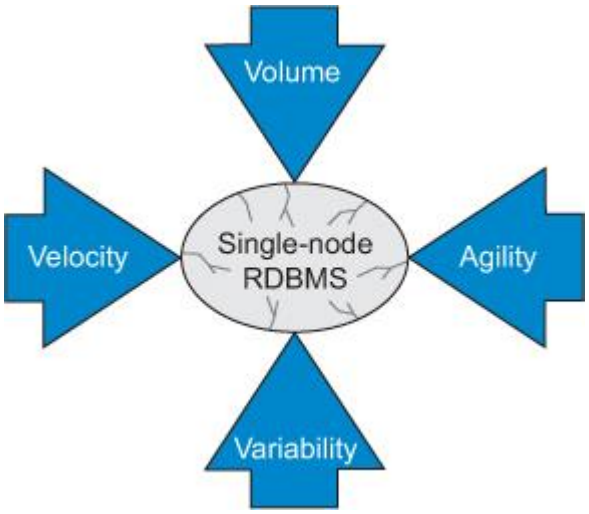
1.2. NoSQL business drivers

The scientist-philosopher Thomas Kuhn coined the term *paradigm shift* to identify a recurring process he observed in science, where innovative ideas came in bursts and impacted the world in nonlinear ways. We'll use Kuhn's concept of the paradigm shift as a way to think about and explain the NoSQL movement and the changes in thought patterns, architectures, and methods emerging today.

Many organizations supporting single-CPU relational systems have come to a crossroads: the needs of their organizations are changing. Businesses have found value in rapidly capturing and analyzing large amounts of variable data, and making immediate changes in their businesses based on the information they receive.

Figure 1.1 shows how the demands of volume, velocity, variability, and agility play a key role in the emergence of NoSQL solutions. As each of these drivers applies pressure to the single-processor relational model, its foundation becomes less stable and in time no longer meets the organization's needs.

Figure 1.1.



1.2.1. Volume

Without a doubt, the key factor pushing organizations to look at alternatives to their current RDBMSs is a need to query big data using clusters of commodity processors. Until around 2005, performance concerns were resolved by purchasing faster processors. In time, the ability to increase processing speed was no longer an option. As chip density increased, heat could no longer dissipate fast enough without chip overheating. This phenomenon, known as the power wall, forced systems designers to shift their focus from increasing speed on a single chip to using more processors working together. The need to scale out (also known as *horizontal scaling*), rather than scale up (faster processors), moved organizations from serial to parallel processing where data problems are split into separate paths and sent to separate processors to divide and conquer the work.

1.2.2. Velocity

Though big data problems are a consideration for many organizations moving away from RDBMSs, the ability of a single processor system to rapidly read and write data is also key. Many single-processor RDBMSs are unable to keep up with the demands of real-time inserts and online queries to the database made by public-facing websites. RDBMSs frequently index many columns of every new row, a process which decreases system performance. When single-processor RDBMSs are used as a back end to a web store front, the random bursts in web traffic slow down response for everyone, and tuning these systems can be costly when both high read and write throughput is desired.

1.2.3. Variability

Companies that want to capture and report on exception data struggle when attempting to use rigid database schema

structures imposed by RDBMSs. For example, if a business unit wants to capture a few custom fields for a particular customer, all customer rows within the database need to store this information even though it doesn't apply. Adding new columns to an RDBMS requires the system be shut down and ALTER TABLE commands to be run. When a database is large, this process can impact system availability, costing time and money.

1.2.4. Agility

The most complex part of building applications using RDBMSs is the process of putting data into and getting data out of the database. If your data has nested and repeated subgroups of data structures, you need to include an object-relational mapping layer. The responsibility of this layer is to generate the correct combination of INSERT, UPDATE, DELETE, and SELECT SQL statements to move object data to and from the RDBMS persistence layer. This process isn't simple and is associated with the largest barrier to rapid change when developing new or modifying existing applications.

1.3. NoSQL case studies

1.3.1. Case study: LiveJournal's Memcache

Background: LiveJournal faced performance issues due to growing traffic and increased demand on their web servers. Each server had its own RAM, leading to inefficient use of memory and repeated SQL queries.
Solution: Engineers at LiveJournal developed Memcache to improve performance. They created a system to cache the results of frequently used database queries in RAM. Each query was assigned a unique "signature" or hash, allowing servers to check if another server had already cached the query result. This reduced the load on the SQL database by avoiding redundant queries.