The field you choose as your hashed shard key should have a good cardinality, or large number of different values. Hashed keys work well with fields that increase monotonically like *ObjectId* values or timestamps.

If you shard an empty collection using a hashed shard key, MongoDB will automatically create and migrate chunks so that each shard has two chunks. You can control how many chunks MongoDB will create with the `numInitialChunks` parameter to `shardCollection` or by manually creating chunks on the empty collection using the `split` command.

To shard a collection using a hashed shard key, see *Shard a Collection Using a Hashed Shard Key* (page 696).

Tip

MongoDB automatically computes the hashes when resolving queries using hashed indexes. Applications do **not** need to compute hashes.

Impacts of Shard Keys on Cluster Operations

The shard key affects write and query performance by determining how the MongoDB partitions data in the cluster and how effectively the `mongos` instances can direct operations to the cluster. Consider the following operational impacts of shard key selection:

Write Scaling   Some possible shard keys will allow your application to take advantage of the increased write capacity that the cluster can provide, while others do not. Consider the following example where you shard by the values of the default *_id* field, which is *ObjectId*.

MongoDB generates `ObjectId` values upon document creation to produce a unique identifier for the object. However, the most significant bits of data in this value represent a time stamp, which means that they increment in a regular and predictable pattern. Even though this value has *high cardinality* (page 696), when using this, *any date, or other monotonically increasing number* as the shard key, all insert operations will be storing data into a single chunk, and therefore, a single shard. As a result, the write capacity of this shard will define the effective write capacity of the cluster.

A shard key that increases monotonically will not hinder performance if you have a very low insert rate, or if most of your write operations are `update()` operations distributed through your entire data set. Generally, choose shard keys that have *both* high cardinality and will distribute write operations across the *entire cluster*.

Typically, a computed shard key that has some amount of "randomness," such as ones that include a cryptographic hash (i.e. MD5 or SHA1) of other content in the document, will allow the cluster to scale write operations. However, random shard keys do not typically provide *query isolation* (page 676), which is another important characteristic of shard keys.

New in version 2.4: MongoDB makes it possible to shard a collection on a hashed index. This can greatly improve write scaling. See *Shard a Collection Using a Hashed Shard Key* (page 696).

Querying   The `mongos` provides an interface for applications to interact with sharded clusters that hides the complexity of *data partitioning*. A `mongos` receives queries from applications, and uses metadata from the *config server* (page 670), to route queries to the `mongod` instances with the appropriate data. While the `mongos` succeeds in making all querying operational in sharded environments, the *shard key* you select can have a profound affect on query performance.

See also:

The *Sharded Cluster Query Routing* (page 678) and *config server* (page 670) sections for a more general overview of querying in sharded environments.