Summary

Data management is essential for most Web applications. The requirements these applications put on their data store are two-fold: first, the data store should provide advanced functionalities such as strong data consistency and complex-query support to improve programmer efficiency in application development; second, the data store should provide properties such as scalability, elasticity and high availability to maintain reasonable performance under arbitrary workload while keeping operational costs under control. This thesis addresses the question: is it possible to build a data store that accommodates all these requirements simultaneously?

Web application developers have access to two main families of data stores: relational databases and NoSQL data stores. However, none of them meet all the aforementioned requirements. Relational databases support ACID transactions and complex queries, but implementing scalable and elastic applications using them requires considerable efforts from application programmers. In contrast, NoSQL data stores provide good properties of scalability and high availability, but lack support for strong data consistency and complex queries. Based on this observation, this thesis explores two different approaches to build a data store which provides all these properties.

First, we study to which extent Web applications based on relational databases can be made elastic and scalable. Second, we explore how one may extend the existing NoSQL data stores with high-level database functionalities, such that their properties of scalability and elasticity are not compromised.

The Relational-Database Approach

In Chapter 3, we show how one may scale Web applications using relational databases, while retaining their transactional properties. We present a systematic approach to denormalize data into a number of independent data services, each of which having exclusive access to its private data store. This restructuring by itself does not lead to linear scalability directly. However, each of the data services has reduced workload complexity, which allows for a more effective application of the optimization techniques such as database replication, query caching and horizontal data partitioning, thus leading to significantly better scalability. For example, read-only data services can be scaled simply by database replication, while update-intensive data services can be scaled more effectively by horizontal data partitioning.
ing. Importantly, the restructuring does not imply any loss in terms of transactional or consistency properties.

The evaluation shows that the restructured application achieves linear scalability with increasing number of database instances. However, this approach requires significant manual efforts in restructuring the application, and in implementing data partitioning for scaling update-intensive data services. This formed the motivation for us to turn to NoSQL data stores which are inherently scalable and partition data automatically.

The NoSQL-Data-Store Approach

In Chapter 4 and 5, we explore how one may extend the existing NoSQL data stores with support of ACID transactions and complex queries, such that their properties of scalability and elasticity are not compromised.

We implement these missing features of NoSQL data stores in a middleware layer, called CloudTPS which sits between the application and its data store. Our prototype creates a temporary copy of the application data in the memory of its participant machines. All the added functionalities, such as transactions and join queries, operate directly on this copy of the data. All updates are checkpointed back to the underlying data store in a lazy fashion such that users observe strong ACID properties even in the case of machine failures or network partitions. CloudTPS follows the system model of typical NoSQL data stores, which automatically manages data partitioning across any number of machines. CloudTPS also replicates data items to a specified number of machines. When encountering machine failures or network partitions, CloudTPS can recover automatically without compromising data consistency. In Chapter 4, we describe the implementation of transactional functionalities as well as system optimizations such as memory management to support large volumes of data. Chapter 5 extends the transaction commit protocol to allow the system to support complex queries such as join and secondary-key queries. To our knowledge, CloudTPS is currently the only NoSQL data store capable of supporting such queries without compromising scalability or strong data consistency.