ABSTRACT

A central promise of cloud services is elastic, on-demand provisioning. The provisioning of data on temporarily available nodes is what makes elastic database services a hard problem. The essential task that enables elastic data services is bringing a node and its data up-to-date. Strategies for high availability do not satisfy the need in this context because they bring nodes online and up-to-date by repeating history, e.g., by log shipping. Nodes must become up-to-date and useful for query processing incrementally by key range. What is wanted is a technique such that in a newly added node, during each short period of time, an additional small key range becomes up-to-date, until eventually the entire dataset becomes up-to-date and useful for query processing, with overall update performance comparable to a traditional high-availability strategy that carries the entire dataset forward without regard to key ranges. Even without the entire dataset being available, the node is productive and participates in query processing tasks. The authors’ proposed solution relies on techniques from partitioned B-trees, adaptive merging, deferred maintenance of secondary indexes and of materialized views, and query optimization using materialized views. The paper introduces a family of maintenance strategies for temporarily available copies, the space of possible query execution plans and their cost functions, as well as appropriate query optimization techniques.

Keywords: Adaptive Merging, Cloud Services, Data Management, Key Range, Nodes, Partitioned B-Trees, Query Optimization Techniques, Query Processing

DOI: 10.4018/jdwm.2013040101
INTRODUCTION

The central promise of cloud services, as opposed to traditional application deployment with fixed assignments of applications to execution resources, is elasticity (Mell & Grance, 2011) and thus flexibility: as demand for specific services grows or shrinks, appropriate resources can be allocated to optimize efficient delivery of all services. For stateless services, this requires, essentially, to start appropriate software on appropriate machines and to adapt network and request routing. For “stateful” services, however, in particular for database services, growing and shrinking the set of servers dedicated to a specific service requires re-partitioning of those data and moving them as appropriate.

Moving a large data set is slow. For example, a modern disk drive with 2 TB storage capacity and transfer bandwidth of 200 MB/s requires 10,000 seconds or about 3 hours just to read all data. A service for which adding or removing nodes from a service takes hours provides only an unsatisfactory degree of elasticity, in particular if competitors, using different techniques, can add or drop nodes in minutes.

We believe that the speed of elastic adaption is a crucial quality of a cloud service. We further believe that solving a minimized prototypical instance of the problem – adding a single node to a single existing node and bringing the new node up-to-date efficiently – is sufficient to explore alternative techniques. Therefore, this minimized instance of this elasticity is the major first step in this paper. Ultimately, once a single node can be added to a cloud database system, scaling the whole system from a small-sized instance to a medium-sized instance to a large or extra large sized instance can be accomplished. However, optimization techniques for large-scale growing or shrinking of the system by many nodes at a time are not further investigated and may be considered in the future.

Our contribution is a technique that permits bringing a new node up-to-date in small, useful steps. In other words, the new node can become useful for query processing (and thus aid scalability of the cloud service) almost instantly and incrementally during the update process. Moreover, the overall duration of the update process is similar to that of traditional log shipping techniques, which bring a new node up-to-date all at once but may take hours. Finally, we describe query optimization techniques, specifically query execution plans and their cost functions, for nodes that are partially updated and continue incremental updates between query optimization and query execution as well as during query execution.

The remainder of this paper is structured as follows. The next section revisits prior work and gives an overview of the relevant techniques, which are combined in novel ways in our solution. The following section states the general problem and explains how it is reduced to a minimal scenario, followed by another section that illustrates our approach to support function shipping and data shipping to a single node that joins a cluster of a database system. The penultimate section explains how the solution for the single-node scenario can be adapted to the general case. We also show several benefits that can be reaped from the elasticity thus achieved. The last section gives conclusions.

RELATED WORK

There are numerous existing techniques for failover or high availability (HA). Typical examples are cluster failover or migration of virtual machines (VMs). A common underlying assumption is that the respective systems are online and either primary active or in standby state. Every commercial database system supports a number of failover approaches. IBM’s HACMP is one such example (Abderrazag, 2008). General database cluster approaches are described in Akal et al. (2002) and Camargos et al. (2007). With the advent of virtualization approaches to VM migration gained momentum those are based on a shared disk assumption but allow for easy workload dependent VM instance migration. Minhas et al. (2011) propose an improved version of a transparent high-availability solution for database management systems