Schism: Workload-Driven Partitioning and Replication

Curino et al., VLDB 2010
Presented By: Brad Glasbergen
OLTP Workloads

- Short-lived transactions
- Touch few data items
- Write-heavy
Scaling OLTP Databases

Overloaded!
Scaling **Out** OLTP Databases

![Diagram showing two databases (A and B) and users with orders being distributed between them.](attachment:image.png)
Distributed Transactions

Commit at both or not at all!
Two-Phase Commit

Run Transaction Logic

A = A - 1
B = B - 1
Two-Phase Commit (Phase 1)

Can we commit?
Two-Phase Commit (Phase 1)

Can we commit?

A OK
B OK
Two-Phase Commit (Phase 2)

Commit!
Two-Phase Commit (Phase 2)

Commit!
2PC Overheads

Lock(A,B)

Blocked!
2PC Overheads
Avoiding 2PC

Move B
Avoiding 2PC

Overloaded?

A,B
Objectives

1. Minimize distributed transactions
2. Roughly balance load/data

Similar to Graph Partitioning problem!
Minimize Edge-cuts subject to imbalance factor
K-Way Graph Partitioning
K-Way Graph Partitioning

Each partition is at most twice the weight of another.
K-Way Graph Partitioning

Each partition is at most twice the weight of another
K-Way Graph Partitioning

Each partition is at most twice the weight of another
K-Way Graph Partitioning

Each partition is at most twice the weight of another
Mapping Partitioning to Graphs

Determine tuple access patterns and frequencies!
Mapping Partitioning to Graphs

Accessed tuples become nodes

A

B

C
Mapping Partitioning to Graphs

Edges:
Frequency of tuples accessed together
Decide balance factor

Each partition is at most twice the weight of another.
Mapping Partitioning to Graphs

Each partition is at most twice the weight of another
Incorporating Replication

Diagram:
- Node A
- Node B
- Node C
- Edges:
  - A to B: Label 1
  - A to C: Label 2
  - B to C: Label 1
Incorporating Replication
Incorporating Replication

Internal edges: update count!
Incorporating Replication

Internal edges: update count!
Explaining the Partitioning

Yes

ID <= 2

ID = 1

{0,1}

No

ID <= 4

0 0 1

Prefer simple explanations
Schism Partitionings

- **YCSB-Default**: Hash Partitioning
- **YCSB-Range**: Range Partitioning
- **TPC-C**: Hash on warehouse, replicate items table
- **TPC-E**: Lookup table, no good known manual partitioning
- **Epinions**: Lookup table, beats manual partitioning

Longest Partitioning: 12 minutes!
SWORD: workload-aware data placement and replica selection for cloud data management systems

K. Aishwarya Kumar - Abdul Qureshi - Amol Deshpande - Samir Khuller

Abstract Cloud computing is increasingly being seen as a way to reduce infrastructure costs and add elasticity, and is being used by a wide range of organizations. Cloud data management systems today need to serve a range of different workloads, from analytical read-heavy workloads to transactional (OLTP) workloads. For both, the service providers and the users, it is critical to minimize the consumption of resources like CPU, memory, communication bandwidth, and energy, without compromising on service-level agreements (SLAs). In this article, we develop a workload-aware data placement and replication approach, called SWORD, for minimizing resource consumption in such an environment. Specifically, we monitor and model the expected workload in a hypergraph and develop partitioning techniques that minimize the average query span, i.e., the average number of machines involved in the execution of a query on a transaction. We empirically justify the use of query span as the metric to optimize, for both analytical and transactional workloads, and develop a series of replication and data placement algorithms by drawing connections to several well-studied graph-theoretic concepts. We introduce a suite of novel techniques to achieve high scalability by restricting the overhead of partitioning and query routing. To deal with workload changes, we propose an intelligent load-partitioning technique that modifies data placement in small steps without interrupting the system.

Keywords Cloud data management - Hypergraph partitioning - Data placement - Replication - Resource minimization - Scalability

1 Introduction

Cloud computing is increasingly embracing a wide range of organizations because of its promise to reduce infrastructure costs and provide elasticity on demand. This has led to a proliferation of cloud-based data management systems to enable such services, and data centers to provide the computational infrastructure for these. Cloud data management systems today need to serve a range of different workloads. These include mostly read-only analytical workloads that need to process large volumes of data in a resource-efficient manner, as well as transactional OLTP-style workloads that need to support high throughput with low latencies. For both the service provider and the users, it is crucial to minimize the total resource consumption in executing these workloads, without compromising on service-level agreements (SLAs). For the service provider, lower resource consumption will enable it to serve a larger number of users without further investment into resources, whereas for the users, lower
ABSTRACT
The advent of affordable, shared-nothing computing systems has made use of parallel database technology in the enterprise feasible. This paper presents a novel approach to enterprise-class OLTP systems that efficiently extends the scale of the OLTP, by (1) eliminating the need for distributed transactions, while concurrently mitigating the effect of temporal skew in both the data distribution and data access; (2) using the design space to include replicated secondary keys, (3) organizing logically distributed physical entry points, and (4) using statistical database models to effectively use the OLTP's efficiency for in-database access to the application's data. In this paper, we present a new online load partitioning approach, called Clay, that supports both two-phase protocol and more complex "generic" schemas with advanced load key adaptation. This approach dynamically selects block sizes and targets of migration among servers during operation, using an efficient, incremental adaptive hash partitioning algorithm. We compare our approach to existing techniques. The results show that Clay achieves the load balance and reduces the amount of data transferred signficantly faster than existing approaches.

1. INTRODUCTION

Many OLTP applications are subject to unpredictable and volatile demand. In this environment, scalability and efficiency of services as well as services themselves are subject to change. As a result, it is important that a good OLTP be resilient and scalable. In this paper, we show that our approach is characterized by a feature in the OLTP's efficiency for in-database access to the application's data.

1. INTRODUCTION

The problem of scalability in on-demand applications is well known. OLTP systems are highly sensitive to changes in performance and to changes in the way data is distributed and accessed. This problem is exacerbated when data is distributed across multiple servers. In these systems, the database must scale across multiple servers in order for the system to scale. In this paper, we present a novel approach to enterprise-class OLTP systems that efficiently extends the scale of the OLTP, by (1) eliminating the need for distributed transactions, while concurrently mitigating the effect of temporal skew in both the data distribution and data access; (2) using the design space to include replicated secondary keys, (3) organizing logically distributed physical entry points, and (4) using statistical database models to effectively use the OLTP's efficiency for in-database access to the application's data.

In this paper, we present a novel approach to enterprise-class OLTP systems that efficiently extends the scale of the OLTP, by (1) eliminating the need for distributed transactions, while concurrently mitigating the effect of temporal skew in both the data distribution and data access; (2) using the design space to include replicated secondary keys, (3) organizing logically distributed physical entry points, and (4) using statistical database models to effectively use the OLTP's efficiency for in-database access to the application's data.
Discussion Points

- Schism is offline/periodic. How important is online partitioning, *really*?
- Would an OLAP workload change how we partition? Consider parallel query execution.