Scalable Data Management in the Cloud: Research Challenges & New Opportunities

Divyakant Agrawal
Department of Computer Science
UC Santa Barbara

Collaborators: Amr El Abbadi, Sudipto Das, Aaron Elmore
…Cloud Computing? What are you talking about? Cloud Computing is nothing but a computer attached to a network.

-- Larry Ellison, Excerpts from an interview
Outline

- Infrastructure Disruption
  - Enterprise owned ➔ Commodity shared infrastructures
  - Disruptive transformations: Software and Service Infrastructure

- Clouded Data Management
  - State of the Art lacks “cloud” features
  - Transactional systems (Application Development)
  - Decision support system (Data Analysis)

- Cloudy Application Landscape

- Gen-next Data Management (UCSB)
  - Design Principles
  - Data Fusion and Fission
  - Elasticity
WEB is replacing the Desktop
Cloud Computing: Why Now?

- Experience with very large datacenters
  - Unprecedented economies of scale
  - Transfer of risk

- Technology factors
  - Pervasive broadband Internet
  - Maturity in Virtualization Technology

- Business factors
  - Minimal capital expenditure
  - Pay-as-you-go billing model
Economics of Data Centers

- Risk of over-provisioning: underutilization

Money & Time Questions:
1. How much?
2. How Long?

Static data center
Economics of Internet Users

- Heavy penalty for under-provisioning

12/9/2010
Economics of Cloud Computing

- Pay by use instead of provisioning for peak capacity
Cloud Computing Spectrum

- Infrastructure-as-a-Service (IaaS)
- Platform-as-a-Service (PaaS)
- Software-as-a-Service (SaaS)

Lower-level, Less management

EC2

Higher-level, More management

Azure AppEngine Force.com
The Big Picture

- Unlike the earlier attempts:
  - Distributed Computing, Distributed Databases, Grid Computing

- Cloud Computing is REAL:
  - Organic growth: Google, Yahoo, Microsoft, and Amazon
  - IT Infrastructure Automation
  - Economies-of-scale
  - Fault-tolerance: automatically deal with failures
  - Time-to-market: no upfront investment
Cloud Reality

- Facebook Generation of Application Developers

- Animoto.com:
  - Started with 50 servers on Amazon EC2
  - Growth of 25,000 users/hour
  - Needed to scale to 3,500 servers in 2 days (RightScale@SantaBarbara)

- Many similar stories:
  - RightScale
  - Joyent
  - ...

COMAD'2010 Keynote Address 12/9/2010
Outline

- Infrastructure Disruption
  - Enterprise owned ➔ Commodity shared infrastructures
  - Disruptive transformations

- Clouded Data Management
  - State of the Art lacks “cloud” features
  - Transactional systems
  - Decision support system

- Cloudy Application Landscape

- Gen-next Data Management systems
  - Design Principles
  - Data Fusion and Fission
  - Elasticity
Current State

- Most enterprise solutions are based on RDBMS technology.

- Significant Operational Challenges:
  - Provisioning for Peak Demand
  - Resource under-utilization
  - Capacity planning: too many variables
  - Storage management: a massive challenge
  - System upgrades: extremely time-consuming
  - Complex mine-field of software and hardware licensing

- Unproductive use of people-resources from a company’s perspective
Scaling in the Cloud

Database becomes the Scalability Bottleneck
Cannot leverage elasticity
Scaling in the Cloud

- Client Site
- HAProxy (Load Balancer)
- Elastic IP
- Apache + App Server
- MySQL Master DB
- Replication
- MySQL Slave DB
Scaling in the Cloud

Client Site → HAProxy (Load Balancer) → Elastic IP

Apache + App Server

Scalable and Elastic
But limited consistency and operational flexibility
Cloud Computing Desiderata

- Scalability
- Elasticity
- Fault tolerance
- Self Manageability
- Sacrifice consistency?
  - Foregone Conclusion!!!
Outline

- Infrastructure Disruption
  - Enterprise owned => Commodity shared infrastructures
  - Disruptive transformations

- Clouded Data Management
  - State of the Art lacks “cloud” features
  - Transactional systems
  - Decision support system

- Cloudy Application Landscape

- Gen-next Data Management(UCSB)
  - Design Principles
  - Data Fusion and Fission
  - Elasticity
Internet Chatter

The Death of Row-Oriented RDBMS Technology. « Kevin Closson's ...
Sep 13, 2007 ... 10 Responses to "The Death of Row-Oriented RDBMS Technology." Feed for
This Entry Trackback Address. 1 Noons September 13, 2007 at 4:01 am ...
kevinclosson.wordpress.com/2007/09/13/the-death-of-row-oriented-rdbms-technology/ - 34k -
Cached - Similar pages

RDBMS: Reports of Its Death Exaggerated: Beyond Search
Database Doomed?" is an interesting article. ...
arnoldit.com/wordpress/2009/02/14/rdbms-reports-of-its-death-exaggerated/ - 33k -
Cached - Similar pages

Web 3.0 And The Decline of the RDBMS | HaveMacWillBlog (aka Robin ...
Feb 1, 2009 ... The Death of RDBMS. Kingsley has also been pursuing a theme that I have been
espousing in recent times, which is that the age of the RDBMS ...
havemacwillblog.com/2009/02/01/web-3-0-an-evolving-debate/ - 46k - Cached - Similar pages

Why does everything suck?: The Death of the Relational Database
The construction of RDBMS is a result of NOT finding this structure to ... The "why relational
databases suck" topic is pretty well beaten to death by ...
whydoeseverythingsuck.com/2008/02/death-of-relational-database.html - 182k -
Cached - Similar pages

Oracle WTF: Death By Furniture
Death By Furniture. According to www.identifiers.org, there are two classes ... Rename the
table or a column – if you can’t, then the RDBMS is Code Class. ...
oracle-wtf.blogspot.com/2008/10/death-by-furniture_12.html - 30k - Cached - Similar pages

Gavin defends RDBMS and Ted rebukes [kirk.blog-city.com]
Gavin defends RDBMS and Ted rebukes. "H E " email, posted Monday, 26 June 2007 ...
“If you want vast, on-demand scalability, you need a non-relational database.” Since scalability requirements:
- Can change very quickly and,
- Can grow very rapidly.

Difficult to manage with a single in-house RDBMS server.

Although RDBMS scale well:
- When limited to a single node (scale-up NOT scale-out).
- Overwhelming complexity to scale on multiple servers.
public void confirm_friend_request(user1, user2) {
    begin_transaction();
    update_friend_list(user1, user2, status.confirmed);
    //user1@Palo Alto Data Center
    update_friend_list(user2, user1, status.confirmed);
    //user2 @London Data Center
    end_transaction();
}
public void confirm_friend_request_A(user1, user2) {
    try {
        update_friend_list(user1, user2, status.confirmed); //palo alto
    } catch (exceptionone) {
        report_error(e); return;
    }
    try {
        update_friend_list(user2, user1, status.confirmed); //london
    } catch (exceptionone) {
        revert_friend_list(user1, user2);
        report_error(e); return;
    }
}
public void confirm_friend_request_B(user1, user2) {
    try {
        update_friend_list(user1, user2, status.confirmed); //palo alto
    } catch (exception e) {
        report_error(e);
        add_to_retry_queue(operation.updatefriendlist, user1, user2, current_time());
    }
    try {
        update_friend_list(user2, user1, status.confirmed); //london
    } catch (exception e) {
        report_error(e);
        add_to_retry_queue(operation.updatefriendlist, user2, user1, current_time());
    }
}
/* get_friends() method has to reconcile results returned by get_friends() because there may be data inconsistency due to a conflict because a change that was applied from the message queue is contradictory to a subsequent change by the user. In this case, status is a bitflag where all conflicts are merged and it is up to app developer to figure out what to do. */

public list get_friends(user1){
    list actual_friends = new list();
    list friends = get_friends();
    foreach (friend in friends){
        if(friend.status == friendstatus.confirmed){
            // no conflict
            actual_friends.add(friend);
        }else if((friend.status&= friendstatus.confirmed)
                and !(friend.status&=
                friendstatus.deleted)){
            // assume friend is confirmed as long as it wasn’t also deleted
            friend.status = friendstatus.confirmed;
            actual_friends.add(friend);
            update_friends_list(user1, friend, status.confirmed);
        }else{ // assume deleted if there is a conflict with a delete
            update_friends_list(user1, friend, status.deleted)
        }
    }
    return actual_friends;
}
I love eventual consistency but there are some applications that are much easier to implement with strong consistency. Many like eventual consistency because it allows us to scale-out nearly without bound but it does come with a cost in programming model complexity.
Recent work

- Building a database on Amazon S3 [Brantner 2008]
- Consistency Rationing in a Cloud Database [Kraska 2009]
- Unbundling Transactions in the Cloud [Lomet 2009a, 2009b]
- Supporting large number of small applications [Yang 2009]
- ePIC project at NUS [VLDB’2010 papers]
Outline

- Infrastructure Disruption
  - Enterprise owned => Commodity shared infrastructures
  - Disruptive transformations

- Cloudy Application Landscape

- Clouded Data Management
  - State of the Art lacks “cloud” features
  - Transactional systems
  - Decision support system

- Gen-next Data Management (UCSB)
  - Design Principles
  - Data Fusion and Fission
  - Elasticity
Design Principles

- Separate System and Application State
  - System metadata is critical but small
  - Application data has varying needs
  - Separation allows use of different class of protocols

- Limit Application interactions to a single node
  - Allows systems to scale horizontally
  - Graceful degradation during failures
  - Obviate the need for distributed synchronization
Design Principles (contd.)

- **Decouple Ownership from Data Storage**
  - Ownership refers to exclusive read/write access to data
  - Partition ownership – effectively partitions data
  - Decoupling allows light weight ownership transfer

- **Limited distributed synchronization is practical**
  - Maintenance of metadata
  - Provide strong guarantees only for data that needs it
Scalability & Elasticity in the Cloud

- **Data Fusion**
  - Enrich Key Value stores
  - GStore: Efficient Transactional Multi-key access  
    [ACM SOCC’2010]

- **Data Fission**
  - Cloud enabled relational databases
  - ElasTraS: Elastic TranSactional Database  
    [HotClouds2009;Tech. Report’2010]

- Elasticity of Data Services
Data Fusion: GStore
Atomic Multi-key Access

- Key value stores:
  - Atomicity guarantees on single keys
  - Suitable for majority of current web applications

- Many other applications warrant multi-key accesses:
  - Online multi-player games
  - Collaborative applications

- Enrich functionality of the Key value stores [Google AppEngine&MegaStore]
Key Group Abstraction

- Define a granule of on-demand transactional access
- Applications select any set of keys
- Data store provides transactional access to the group
- Non-overlapping groups
Horizontal Partitions of the Keys

A single node gains ownership of all keys in a KeyGroup

Group Formation Phase

COMAD'2010 Keynote Address 12/9/2010
Key Grouping Protocol

- Conceptually akin to “locking”
- Allows collocation of ownership
- Transfer key ownership from “followers” to “leader”

- Guarantee “safe transfer” in the presence of system dynamics:
  - Dynamic migration of data and its control
  - Failures
Implementing GStore

Application Clients

Transactional Multi-Key Access

Grouping Middleware Layer resident on top of a Key-Value Store

Grouping Layer | Transaction Manager
---|---
Key-Value Store Logic

Grouping Layer | Transaction Manager
---|---
Key-Value Store Logic

Grouping Layer | Transaction Manager
---|---
Key-Value Store Logic

Distributed Storage

G-Store

COMAD’2010 Keynote Address   12/9/2010
G-Store Experimental Setup

- Performed in Amazon EC2
- Application benchmark simulating an Online multi-player game
- Cluster size: 10 nodes
- Number of concurrent clients: 20 to 200
- Number of keys in a group: 10 to 100
- Data size: ~1T
- Each node in the cluster: 8 cores, 7G RAM, 1.7T disk
Group Creation Throughput

Group Creation Throughput (100 keys)

- Clientbased-Contiguous
- Clientbased-Random
- Middleware-Contiguous
- Middleware-Random

# of Concurrent Clients

- # Groups created per sec

COMAD'2010 Keynote Address  12/9/2010
Latency for Group Operations

Average Group Operation Latency (100 Opns/100 Keys)

Latency (ms)

# of Concurrent Clients

GStore - Clientbased  GStore - Middleware  HBase

COMAD'2010 Keynote Address
Data Fission: ElasTraS
Elastic Transaction Management

- Designed to make RDBMS cloud-friendly
- Database viewed as a collection of partitions
- Suitable for:
  - Large single tenant database instance
    - Database partitioned at the schema level
    - Multi-tenant database with large number of small databases
      - Each partition is a self contained database
Elastic Transaction Management

- Elastic to deal with workload changes
- Load balance partitions
- Recover from node failures
- Dynamic partition management
- Transactional access to database partitions
ElasTraS Experimental Setup

- Performed in Amazon EC2
- Used TPC-C for evaluation
- Cluster size: 10 to 30 nodes
- Number of concurrent clients: 100 to 1800
- Number of warehouses: 1000 to 3000
- Data size: ~1T
- Each node in the cluster: 8 cores, 7G RAM, 1.7T disk
Latency of Transactions

Latency (10 Nodes, 1000 Warehouses)

Latency (30 Nodes, 3000 Warehouses)

# of Concurrent Clients vs. Latency in ms

NewOrder  Payment  NO + Sleep  Payment + Sleep

COMAD'2010 Keynote Address  12/9/2010
Throughput

Throughput for 10 Nodes, 1000 Warehouses

Throughput for 30 Nodes, 3000 Warehouses

# of Concurrent Clients

Throughput (tpmC)

Throughput

Throughput with Sleep

# of Concurrent Clients

Throughput (tpmC)
Elasticity in the Cloud: Live Data Migration
Elasticity

- A database system built over a pay-per-use infrastructure
  - Infrastructure as a Service for instance

- Scale up and down system size on demand
  - Utilize peaks and troughs in load

- Minimize operating cost while ensuring good performance
Elasticity in the Database Layer
Elasticity in the Database Layer

Capacity expansion to deal with high load –
Guarantee good performance

DBMS
Elasticity in the Database Layer

Consolidation during periods of low load – Cost Minimization

DBMS
Live Database Migration

- All Elasticity induced dynamics in a Live system
- Minimal service interruption for migrating data fragments
  - Minimize operations failing
  - Minimize unavailability window, if any
- Negligible performance impact
- No overhead during normal operation
- Guaranteed safety and correctness
Live Database Migration
Current State – A teaser

- **Shared storage** architecture
  - **Proactive** state migration
    - No need to migrate persistent data
    - Migrate database cache and transaction state proactively
    - Ensures low performance impact

- **Shared nothing** architecture
  - **Reactive** state migration
    - Migrate minimal database state
    - Persistent image migrated asynchronously on demand

- More details to follow in the near future
  - A long presentation in its own merit
Migration in Shared Storage

0. Normal Database Operation

1. Migration Phase
   1a. Begin Migration
   - Snapshot state at $N_{src}$
   - Initialize $C_{migr}$ at $N_{dst}$
   1b. Synchronize and Catch-up
   1c. Atomic Handover Phase
   - Stop serving $C_{migr}$ at $N_{src}$
   - Synchronize remaining state
   - Start serving $C_{migr}$ at $N_{dst}$
   - Resume normal operation

2. Post Migration Phase

Source DBMS Node ($N_{src}$)

Owning DBMS Node

Destination DBMS Node ($N_{dst}$)

Pre Migration Phase

Prepare Phase

Handover Phase

Post Migration Phase
Migration in Shared Nothing

Controll er

Source

Destination

Route

INIT

NORMAL

T_{S1}, \ldots, T_{Sk}

Handover

T_{Sk+1}, \ldots, T_{Sl}

INITIAlize

T_{D1}, \ldots, T_{Dm}

DUAL

On Demand Pull

T_{Dm+1}, \ldots, T_{Dn}

FINISH

Asynchronous Push

T_{Dn+1}, \ldots, T_{Dp}

NORMAL

Migration Modes

COMAD’2010 Keynote Address   12/9/2010
Cloud Computing at UCSB & Santa Barbara

COMAD'2010 Keynote Address  12/9/2010
Research Activities

- Cloud Computing Infrastructures:
  - Rich Wolski, UCSB

- Cloud Programming Models, Applications and Languages:
  - ChadraKrintz, UCSB

- Data Management in Clouds:
  - Divy Agrawal & Amr El Abbadi, UCSB

- Security & Privacy Models in Clouds:
  - Giovanni Vigna & Christopher Kruegel, UCSB
Industrial Start-ups

- **Cloud Computing Infrastructures:**
  - Eucalyptus: Rich Wolski

- **Cloud Computing Management:**
  - RightScale: Thurston von Eicken

- **Application Hosting in the Cloud:**
  - AppFolio: Klaus Schauser
Concluding Remarks

- Data Management for Cloud Computing poses a fundamental challenges:
  - Scalability
  - Reliability
  - Elasticity
  - Payment Model
  - Data Consistency

- Cloud Computing in Emerging Markets:
  - Leveling the playing field in the context of IT

- Finally, the computing substrate will also evolve:
  - Multiple Data Centers
  - Leveraging the Network Edge (beyond content caching)
References

- [Helland 2007] Life Beyond Distributed Transactions: An Apostate’s Opinion by P. Helland, CIDR’07
- [Brantner 2008] Building a Database on S3 by M. Brartner, D. Florescu, D. Graf, D. Kossman, T. Kraska, SIGMOD’08
- [Lomet 2009a] Unbundling Transaction Services in the Cloud by D. Lomet, A. Fekete, G. Weikum, M. Zwilling, CIDR’09
- [Yang 2009] A scalable data platform for a large number of small applications, F. Yang, J. Shanmugasundaram, and R. Yener, CIDR, 2009
An Alternative View