Towards Database Virtualization for Database as a Service

Aaron J. Elmore, Carlo Curino, Divyakant Agrawal, Amr El Abbadi

[aelmore,agrawal,amr] @ cs.ucsb.edu
ccurino @ microsoft.com
CAVEAT:
Representative not exhaustive
Moving to the Cloud

Why cloud? (are you really asking?)

- Economy-of-scale arguments
- Pay-per-use value to customers
Moving to DaaS

Why Database as a Service (DaaS) for tenants?

DB management drama becomes provider’s problem
(Ideally) high level Service Level Agreement (SLAs / SLOs)
Accelerate development lifecycle

Why Database as a Service (DaaS) for providers?

Internalize a high-cost portion of service (admin)
Scale + density + uniformity $\Rightarrow$ lower cost
The illusion we are aiming for...

Tenant’s view

Provider’s view
Traditional DB deployments

Yo... the DB is slow!

$##%@#!
...Try now.

DBAdmin

DB

Developer
What changes in DBaaS?
Commercial DaaS

Tuning issue

Running max-throughput, write-heavy YCSB workload against:
- fully managed DBMS
- manually tuned DBMS

(Same virtualized hardware, same DBMS, different tuning)

Interpretation: default log-configuration is off.

* experiments using OLTPBench: http://oltpbenchmark.com
DaaS: challenges (and agenda)

Multi-tenancy Architectures
SLA/SLO
  Definition
  Enforcement
High Availability
  Replication
  Fault tolerance
Partitioning
  (security/privacy)

Workload Characterization
  Estimation / Prediction
  Resource Attribution
  What if analysis
Resource Management
  Allocation / Balancing
  Tenant Placement
  Admission Control
Migration
  Performance Isolation
Multi-tenancy architectures

“Most common ways to tackle this problem”
Shared Hardware (DB-in-a-VM)

DBMS (tenant1)  DBMS (tenant2)  DBMS (tenant3)  DBMS (tenant4)

Guest OS VM  Guest OS VM  Guest OS VM  Guest OS VM

Hypervisor

Hardware
Shared Process

Database (tenant1) — Database (tenant2) — Database (tenant3) — Database (tenant2)

Database Process

OS

Hardware
Shared Table

Table

Database Process

OS

Hardware

Rows (tenant1)  Rows (tenant2)  Rows (tenant3)  Rows (tenant4)
Trade-off

Shared Hardware
- Strong Isolation (security, performance)
- Mechanics (High Availability, Migration)

Shared Process
- Sharing and coordination resource consumption (MEM/CPU/Disk IOPs)

Shared Table
- Amortize metadata overheads
Multi-tenancy Architectures

Shared Hardware

SmartSLA, RemusDB, Amazon RDS

Shared Process

RelationalCloud, CloudDB, SQLAzure, Delphi, Y! cidr2009
(shared storage) ElasTras, DAX

Shared Table

Force.com, Jacobs/Aulbach
Shared Hardware (DB-in-a-VM)

“Reusing/Specializing VM technologies for DaaS”
Commercial offering: Amazon RDS

Amazon RDS

Provides pre-configured DBMS (MySQL/Oracle/SQLServer)

Addresses much of provisioning issues

Strong Isolation / catch-all configuration
SmartSLA

[Xiong et al. ICDE 2011]

Focus

Leverage VM-based mechanisms
Deliver DB-level SLAs

Key Contribution

SLA violation vs Resource modeling
Actuation of VM-based mechanisms (cpu, ram, replication)
SmartSLA

Key mechanism

Decompose problem in:
ML-based model of resource / SLA-penalty
Allocation of resource + replication
Estimating SLA violation cost and Allocation

ML Modeling

Build a Map of space
(simple ML/features)

Allocation algorithm

Explore allocation space
Models infrastructure cost for replication
Models cost of increasing replication
Focus

High Availability via VM replication
OLTP-compatible performance

Key Contributions

Reuse of mature VM technology (*pro of Shared Hardware*)
Smart DB-specific tricks to improve performance
REMUS

Leverage Xen VM-replication

Snapshots the VM state every few tens of ms
Delays network and disk writes until next checkpoint (consistent)
Fail-over to secondary and restart from latest checkpoint

Problems

DBMS bufferpool changes too fast (large deltas to checkpoint)
Latency overhead is high for OLTP
REMUSDB: DB-specific optimizations

Avoid checkpointing “clean” pages
  no checkpoint for clean pages
  bookkeeping so that secondary fetch from disk if needed

Limit network delay to Commit/Abort
  Leverage transactional semantics
  “delay” only Commit/Abort messages

Reduce impact on throughput
  32% goes down to about 10%
Design mismatch

DBMS were designed to make full use of dedicated machines
Aggressively consume idle resources (especially IOPs)
[Curino et al. VLDB 2010]
Shared Process

“The DBMS knows best”
Commercial offering: SQL Azure [Bernstein et al. ICDE 2011]

SQL Azure

Shared DBMS process, Dedicated database
Shared logging
Modified version of SQL Server
High-availability via quorum of replicas
Support scale-out
  ACID within a row-group
  Read-committed across row-group
ElasTras Architecture
(Shared Storage)

[Das et al. HotCloud 2009]
Scalable and fault tolerant m/t achieved by data layer spanning colos
Use Cassandra for storage tier with single owning DB instance
Leverage DB and quorum semantics for performance
Operation type & R/W/N
Epoch-bounded strong consistency
Shared Process shortcomings
Comparing multi-tenancy (No DBMS is perfect)

* experiments using OLTPBench: http://oltpbenchmark.com
Shared Table

“Extreme multi-tenancy”
DBMSs don’t scale well at the tenant/schema level

![DBMS scaling table]

<table>
<thead>
<tr>
<th></th>
<th>Memory 1 instance</th>
<th>Memory 10,000 instances</th>
<th>Disk 1 instance</th>
<th>Disk 10,000 instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostgreSQL</td>
<td>55</td>
<td>79</td>
<td>4</td>
<td>4,488</td>
</tr>
<tr>
<td>MaxDB</td>
<td>80</td>
<td>80</td>
<td>3</td>
<td>1,168</td>
</tr>
<tr>
<td>Commercial1</td>
<td>171</td>
<td>616</td>
<td>200</td>
<td>414,210</td>
</tr>
<tr>
<td>Commercial2</td>
<td>74</td>
<td>2,061</td>
<td>3</td>
<td>693</td>
</tr>
<tr>
<td>Commercial3</td>
<td>273</td>
<td>359</td>
<td>1</td>
<td>13,630</td>
</tr>
</tbody>
</table>

Table 1. Storage Requirements for Schemas Instances (in megabytes)
Force.com and [Aulbach et al. SIGMOD 2008]

Focus

Target tens of thousands of tenants per server
Partially shared schema (polymorphic SaaS apps)
Deal with schema-level/DBMS scalability limits

Key Contribution

Clever data design, schema mapping / query rewriting
Many variants

Private Table
Extension Table
Universal Table
Pivot Table
Chunk Table
Chunk Folding

SELECT Beds
  FROM Account17
  WHERE Hospital='State'.

(Q1)

SELECT Beds
  FROM (SELECT Str1 as Hospital,
          Int1 as Beds
        FROM Chunk_{int,str}
        WHERE Tenant = 17
        AND Table = 0
        AND Chunk = 1) AS Account17
  WHERE Hospital='State'.

(Q1_{chunk})
Shared Table shortcomings

Focused on extreme multi-tenancy

Middleware-based querying rewriting
Ad-hoc security
Hard to provide performance isolation
Only for small / low-activity tenants
DaaS: challenges (and agenda)

- Multi-tenancy Architectures
- SLA/SLO  
  - Definition
  - Enforcement
- High Availability  
  - Replication
  - Fault tolerance
- Partitioning (security/privacy)

- Workload Characterization
  - Estimation / Prediction
  - Resource Attribution
  - What if analysis
- Resource Management
  - Allocation / Balancing
  - Tenant Placement
  - Admission Control
- Migration
- Performance Isolation
Partitioning

“Chop it and scale it out”
Schism

Positioning

Partitioning for shared-nothing DBMSs (RelationalCloud)

Focus

automatic partitioning of arbitrary schemas (many-to-many)
handle access skew, replication

Key Contributions

Model the problem as graph-partitioning
“Explain” results using decision trees (practical partition functions)
Schism: Graph-based Partitioning
Schism: Graph-based Partitioning

Graph Representation:

- Tuples in the DB are nodes in the graph.
Schism: Graph-based Partitioning

Graph Representation:
tuples in the DB are nodes in the graph
transactions impose edges among the tuples they access
Schism: Graph-based Partitioning

Graph Representation:
- Tuples in the DB are nodes in the graph.
- Transactions impose edges among the tuples they access.
Schism: Graph-based Partitioning

Graph Representation:

- Tuples in the DB are nodes in the graph.
- Transactions impose edges among the tuples they access.
Schism: Graph-based Partitioning

Graph Representation:
- tuples in the DB are nodes in the graph
- transactions impose edges among the tuples they access
Graph Partitioning: find $K$ (close to) balanced partitions of the nodes that minimize the weight of the cut edges (i.e., minimize distributed transactions)
Schism: Graph-based Partitioning

Explanation: *compact, predicate-based representation of the graph-partitioning solution*
Key Contributions

- Repartitioning heuristics
- Scaling to larger problems by pre-processing (hyper)graph
- Greater focus on replication for fault-tolerance
- Use of quorums (not just ROWA)
Horticulture

Focus

Time-varying skew
Handle Store procedures natively

Key Contributions

Schema and workload-driven partitioning
Large neighborhood search (rich cost model + cheap estimation)
Horizontal partitioning + table replication + index replication

[Pavlo et al. 2012]
Horticulture: Cost Model

Both distributed transactions and temporal skew heavily impact performance.

Figure 2: Impact of Distributed Transactions on Throughput

Figure 3: Impact of Temporal Workload Skew on Throughput
Horticulture: Large Neighborhood search

Table Candidate
- Horizontal: C_ID
- Replication: False
- 2ndry Index: {C_ID,C_NM}

Proc Candidate
- Parameter: #1
Throughput comparison (for H-Store)

TPC-C

Skewed

(cost estimate)
lower is better

higher is better

Throughput comparison (for H-Store)
Where are we with partitioning?

Problems we know how to solve:

OLAP (tons of classic work)
OLTP (few recent papers, good grasp on the problem)

More to do:

OLAP-OLTP mixed workloads partitioning
Coordinating replication (and erasure codes) for:
  Performance, Fault-tolerance
Geo-distributed placement/replication
DaaS: challenges (and agenda)

- Multi-tenancy Architectures ✓
- SLA/SLO ✓
  - Definition
  - Enforcement
- High Availability ✓
  - Replication
  - Fault tolerance
- Partitioning ✓
  (security/privacy)

- Workload Characterization
  - Estimation / Prediction
  - Resource Attribution
- What if analysis
- Resource Management
  - Allocation / Balancing
  - Tenant Placement
  - Admission Control
- Migration
  - Performance Isolation
Managing Resource Contention
Finding the Balance

Tenant’s view

Provider’s view
Contention for Resources

Resources are shared, finite, and valuable
Enable “Performance” in a Shared Environment

System needs to isolate the tenants to provide performance when finite resources are shared.
Mechanisms to Enforce Isolation

Hard

Static Provisioning
Resource Allocation
(Dynamic Provisioning)

Soft

Smart Placement
(Admission Control)
DaaS: challenges (and agenda)

Multi-tenancy Architectures ✓
SLA/SLO ✓
  Definition
  Enforcement
High Availability ✓
  Replication
  Fault tolerance
Partitioning ✓
  (security/privacy)

Workload Characterization
  Estimation / Prediction
  Resource Attribution
  What if analysis
Resource Management
  Allocation / Balancing
  Tenant Placement
  Admission Control
Migration
  Performance Isolation
Hard Isolation

“Keeping your word about resource sharing”
SQLVM [CIDR 2013, SIGMOD 2013, VLDB 2014]

Focus

Embedding resource allocation in DBMS kernel.

How to share critical resources required by DB.

How to understand resource allocation.

Key Contributions

Fine grain resource scheduling (CPU, Memory, I/O).

Metering to audit resource promise.
SQLVM Motivation

Query level SLOs are hard.

```
SELECT Product, SUM(Sales) as TotalSales
FROM FactSales F JOIN DimProduct P JOIN DimStates S
ON F.ProdID = P.ProdID and F.StateId = S.StateId
WHERE State = 'Vermont' 'California'
GROUP BY Product
```

Ad-hoc queries add to the challenge.
Resource Governance Mechanism

Tenant is promised reservation of DBMS resources

“VM inside SQL process”
CPU utilization, IOPS, Memory, ...

Resource governance
Fine-grained resource sharing
Novel mechanisms

Metering (auditing)
Monitor actual and promised metrics for tenant
Determine violations
Resource Allocation

CPU
Reservation: CPU utilization (e.g. 10%) for running or runnable tasks

Memory (Buffer Pool)
Reservation: Hit Ratio of workload for given memory size (e.g. 1GB)

Disk I/O: Shaping Traffic
50 IOPS ⇒ one I/O every 20 msec issued
I/O request tagged with deadline. Put into queue
Issue I/Os whose deadline has arrived
Challenges

Metering and auditing resources.

Multi-core CPU scheduling.

Multiple volumes

Indirect and direct work.
Soft Isolation

“Smart placement to mitigate resource contention”
DaaS: challenges (and agenda)

- Multi-tenancy Architectures ✓
- SLA/SLO ✓
  - Definition
  - Enforcement
- High Availability ✓
  - Replication
  - Fault tolerance
- Partitioning ✓
  (security/privacy)

- Workload Characterization
  - Estimation / Prediction
  - Resource Attribution
  - What if analysis
- Resource Management
  - Allocation / Balancing
    - Tenant Placement
  - Admission Control
- Migration
- Performance Isolation ✓
Common Patterns

Understand workloads
- Fixed, Profiled, or Learned
- Isolated vs Consolidated

How workloads combine
- Provided function (oracle)
- Models
- Observations

Find placement
- Incremental
- Bin-packing
- Optimization

Metrics
- Robustness
- Costs (SLA, Operating)
- Performance (TPS, Latency)
Towards Multi-Tenant Performance SLOs

Willis Lang, Srinath Shankar, Jignesh M. Patel, Ajay Kallan
Univ. of Wisconsin and MS Gray Systems Lab

ICDE 2012
Common Patterns

Understand workloads
- Fixed, Profiled, or Learned
- Isolated vs Consolidated

How workloads combine
- Provided function (oracle)
- Models
- Observations

Find placement
- Incremental
- Bin-packing
- Optimization

Metrics
- Robustness
- Costs (SLA, Operating)
- Performance (TPS, Latency)
Towards Multi-Tenant Performance SLOs

Focus

- Different hardware configurations (SKU)
- Multiple tenant performance SLO classes
- Place to meet SLOs and minimize costs

Key Contributions

- Cost aware server consolidation
- Tenant placement optimization framework
Heterogeneous SLO Characterization

Benchmark server to find max degree multi-tenancy for perf objectives

Systematically reduce ‘H’ tenants, steadily increase ‘L’ tenant scheduling until a perf objective fails

Server characterizing function:

- Both perf objectives met
- Some perf objective fails

Slide by Lang et al.
Approach

Assumption

In memory tenant addition is mainly linear.

Solution

One DB instance per SLO throughput class.

(Balancing buffer pool sharing)

Discover frontier

Use solver for ILP formulation to minimizes costs
RTP: Robust Tenant Placement for Elastic In-Memory DB Clusters

Jan Schaffner, Tim Januschowski, Megan Kercher, Tim Kraska, Hasso Plattner, Michael J. Franklin, Dean Jacobs

Hasso Plattner, SAP, UC Berkeley, Brown University

SIGMOD 2013
Common Patterns

Understand workloads
Fixed, Profiled, or Learned
Isolated vs Consolidated

How workloads combine
Provided function (oracle)
Models
Observations

Find placement
Incremental
Bin-packing
Optimization

Metrics
Robustness
Costs (SLA, Operating)
Performance (TPS, Latency)
Robust Tenant Placement

Focus

In memory databases with temporal changes / ethereal DBs
Minimize servers while being robust to failures
Replication with ability to redirect workload

Key Contributions

Incremental algorithms to reduce total costs of ownership
Maintain replication and respect server load.
Migration and existing placement aware solution
Workloads

Workloads are diurnal and short lived bursty tenants.

Workload resource consumption. is univariate and additive

Read heavy workloads
Placing Tenants

Robust to failure (interleaving tenants over bin packing)

Maintain replication

Migration capacity
Solutions

Greedy Heuristics
Meta-heuristics
Exact Solutions

Static and incremental solutions.
Framework

Incremental algorithms follow these steps:

1. Delete un-needed replicas
2. Ensure migration flexibility
3. Create missing replicas
4. Fix overloaded servers
5. Reduce number of active servers
6. Minimize max load
PMAX: Tenant Placement in Multitenant Databases for Profit Maximization

Ziyang Liu, Hakan Hacigümüş, Hyun Jin Moon, Yun Chi, and Wang-Pin Hsiung

NEC Laboratories America

EDBT 2013
Common Patterns

Understand workloads

- Fixed, Profiled, or Learned
- Isolated vs Consolidated

How workloads combine

- Provided function (oracle)
- Models
- Observations

Find placement

- Incremental
- Bin-packing
- Optimization

Metrics

- Robustness
- Costs (SLA, Operating)
- Performance (TPS, Latency)
PMAX

Focus

Latency response SLOs

Workloads are not fixed and vary, history is not available

Profit maximization

Key Contributions

Cost focused placement solution

Bounded approximation algorithms & dynamic prog. solution
Common Patterns

Understand workloads

- Varied arrival rate
- Provided query SLA (over Load = resp. time / arrival)
- Load > 1 = missed SLA

How workloads combine

- Server load = sum tenants load * tenant load factor

Figure 3: Relationship between Average TPC-W Query Processing Time and Number of Tenants on a Server
Placement Formulation

Each server has a operating costs.

Place tenants to minimize costs (occasional violations OK).

Two problem formulations:

Uniform: Fixed arrival rate and SLA
General: Varied arrival and query based SLA
Both reduced to NP-hard
Solution

Best fit heuristic is sub-optimal

Encourage new servers

Use normalized SLA ordering of tenants

Approximation and DP solution
DaaS: challenges (and agenda)

Multi-tenancy Architectures ✔
SLA/SLO ✔
- Definition
- Enforcement
High Availability ✔
- Replication
- Fault tolerance
Partitioning ✔
(security/privacy)

Workload Characterization
- Estimation / Prediction
- Resource Attribution
- What if analysis
Resource Management
- Allocation / Balancing
- Tenant Placement
- Admission Control
Migration
Performance Isolation ✔
Workload-Aware Database Monitoring and Consolidation

Carlo Curino, Evan P.C. Jones, Samuel Madden, and Hari Balakrishnan

MIT

SIGMOD 2011
Common Patterns

Understand workloads
- Fixed, Profiled, or Learned
- Isolated vs Consolidated

How workloads combine
- Provided function (oracle)
- Models
- Observations

Find placement
- Incremental
- Bin-packing
- Optimization

Metrics
- Robustness
- Costs (SLA, Operating)
- Performance (TPS, Latency)
Kairos

Focus

Modeling resource consumption of OLTP workloads
Consolidate workloads

Key Contributions

Method to determine *active working set size*
Model disk I/O for consolidation
Find balanced consolidation plan.
Buffer Pool Gauging for RAM

Databases are greedy
Use ballooning to ID active working set size
Disk Model

With working set in RAM: I/O is flushing and txn logs

Regardless of transaction type, max update throughput of a disk depends primarily on database working set size.

Adding workload metrics holds.
Node Assignment via Optimization

Goal: minimize required machines (leaving headroom), balance load

Problem modeled as:
Mixed-integer non-linear optimization problem

Implemented in DIRECT non-linear solver; several tricks to make it go fast

Constraint Violation Penalty
(workloads assignments, max resources is violated)

Global Minimum
(minimum possible $K$, and balanced load for $K$ servers)

Local Minima
(balanced load for $K$ servers)

Load assigned to one of the servers
DaaS: challenges (and agenda)

Multi-tenancy Architectures ✔
SLA/SLO ✔
  Definition
  Enforcement
High Availability ✔
  Replication
  Fault tolerance
Partitioning ✔
(security/privacy)

Workload Characterization
  Estimation / Prediction
  Resource Attribution
  What if analysis
Resource Management
  Allocation / Balancing
  Tenant Placement
  Admission Control
Migration
Performance Isolation ✔
Performance and resource modeling in highly-concurrent OLTP workloads

Barzan Mozafari, Carlo Curino, Alekh Jindal, Samuel Madden

MIT, MS CSIL

SIGMOD 2013
Common Patterns

Understand workloads

- Fixed, Profiled, or **Learned**
- Isolated vs **Consolidated**

How workloads combine

- Provided function (oracle)
- **Models**
- Observations

Find placement

- Incremental
- Bin-packing
- Optimization

Metrics

- Robustness
- Costs (SLA, Operating)
- Performance (TPS, Latency)
DBSeer

Focus

Attribute resource consumption to txn classes (and tenants)
Attribute at runtime in consolidated process
Build models of various DB resources

Key Contributions

Models for disk I/O, locks, throughput, etc
Attribute resources to tenants.
Ability for DBAs to play what-if
DBSeer From 10000 ft

(1) Input (logs)

- SQL logs
  - time, connection, sql_statement
  - 10:23:13.30, C1-BEGIN TRANSACTION

- DBMS logs
  - time window, # pages, # writes, # locks
  - 10:23:13, 001, 2000, 0.12

- OS logs
  - time, CPU, RAM, IOs
  - 10:23:13, 89, 2005, 119
  - 10:23:14, 85, 2010, 53
  - 10:23:15, 87, 2007, 140

(2) Preprocessing / Clustering

(3) Modeling

- disk model
- cpu model
- lock model
- ... model

DB Admin

Reconfigure / Tune

What-if questions
Transaction Clustering

Problem: Different transaction have different access patterns

SQL Logs
- time connection sql stmt
- 1:92 C1 BEGIN TRANSACTION
- 1:93 C2 SELECT * FROM ...

1. Extract features of each transaction
   - number of rows read/written to each table
2. Run DBSCAN clustering algorithm
Predicting Disk I/O

**Disk Reads** = Cache miss rate * # logical reads

**Disk Writes** = log IO + data IO

  - **Log IO (sequential):** redo logs
  - **Data IO (random):** dirty pages
    - due to log reclamation
    - due to page evictions (buffer pool misses)

**Key Observation:**

  # dirty pages flushed = # new pages getting dirtied

  **Predict # of dirty pages**
Other Components of DBSeer

Clustering transactions
Disk Writes
RAM/Disk Reads
  Predicting expected cache-miss rate
Lock Contention
  Queuing theory techniques
Network, CPU, Logical I/O, Logging
  Linear regression
Max Throughput
  Finding the bottleneck resource
DaaS: challenges (and agenda)

- Multi-tenancy Architectures
- SLA/SLO
  - Definition
  - Enforcement
- High Availability
  - Replication
  - Fault tolerance
- Partitioning (security/privacy)
- Workload Characterization
  - Estimation / Prediction
  - Resource Attribution
  - What if analysis
- Resource Management
  - Allocation / Balancing
  - Tenant Placement
- Admission Control
- Migration
- Performance Isolation
Characterizing tenant behavior for placement and crisis mitigation in multitenant DBMSs

Aaron J. Elmore, Sudipto Das, Alexander Pucher, Divyakant Agrawal, Amr El Abbadi, Xifeng Yan

UC Santa Barbara, MSR

SIGMOD 2013
Common Patterns

Understand workloads
- Fixed, Profiled, or Learned
- Isolated vs Consolidated

How workloads combine
- Provided function (oracle)
- Models
- Observations

Find placement
- Incremental
- Bin-packing
- Optimization

Metrics
- Robustness
- Costs (SLA, Operating)
- Performance (TPS, Latency)
Pythia

Focus

Tenant workloads are unknown, disk-based, and dynamic
Use supervised learning to model tenants and colocation
Leverage models to resolve performance crisis

Key Contributions

Method for empirically learning how tenant classes collocate
End to end framework for tenant placement
Tenant Model

Want to construct a tenant model which given a vector of database attributes provides a tenant class (or label).

Tenant based on database agnostic attributes
- (TPS, cache hit %, buffer pool size, write %, etc)

Easily available and available after consolidation

Correlates to tenants’ behavior and performance requirements
Describe Resource Consumption

Tenant labels should describe resource consumption. For example, we are concerned with: Disk and CPU. Use colored shapes as example classes:

- Disk Heavy
- Disk Medium
- Disk Light
- CPU Heavy
- CPU Light

Train a function $T$: set of tenant / DB attributes $\rightarrow$ class

$T(\begin{array}{c}
\text{Feature 1} \\
\text{Feature 2} \\
\text{Feature 3}
\end{array}) = \triangle$
Learn which classes collocate well

Want to see how a node is performing

- Under: ✓ +
- Good: ✓
- Over: ×

Boundaries set by administrator.
Uses resources and latency SLOs.
Control over consolidation.

Incrementally learned through observation
Things Don’t Always Go To Plan

Single tenant in percentile latency causes a node violation.

Use node model to identify set of tenants to remove and identify destinations to receive tenants.

How to identify which tenants and destination nodes?
Searching for a solution

Implemented as a hill-climbing algorithm

Each step is a migration

Evaluate the sum of: each nodes “over”-ness * # tenants
DaaS: challenges (and agenda)

- Multi-tenancy Architectures
- SLA/SLO ✔
- Definition
- Enforcement
- High Availability ✔
- Replication
- Fault tolerance
- Partitioning ✔
- (security/privacy)

- Workload Characterization
  - Estimation / Prediction
  - Resource Attribution
  - What if analysis ✔

- Resource Management
  - Allocation / Balancing
  - Tenant Placement ✔
  - Admission Control

- Migration
- Performance Isolation ✔
Migration for Load Balancing
Migration Forms

Want to move a database between servers

Naïve: *Stop-and-copy*

Improvement: *Flush-and-copy*

Replication based: *Synchronous*

Ideal: *Live Migration*
Migration Goals

Downtime

Service Interruption

Migration Overhead

Time to Complete
Albatross

Focus

Live migration in a shared storage transactional DB
Migration TM state and cache

Key Contributions

First live migration for shared storage.
Minimal strain on destination
Live Migration for Shared Storage

DBMS Node

Tenant/DB Partition

Cached DB State

Transaction State

Source

Destination

Persistent Image

Cached DB State

Transaction State
Albatross Live Migration

Ownership

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initiate Migration</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>▪ Snapshot cache at $N_{src}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>▪ Initialize tenant at $N_{dst}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>▪ $N_{src}$ continues executing transactions</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Synchronize and Catch-up</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>▪ Track changes to DB State at $N_{src}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>▪ Iteratively synchronize state changes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Finalize Migration</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>▪ Stop serving Tenant at $N_{src}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>▪ Synchronize cache</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>▪ Migrate transaction state</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>▪ Transfer ownership to $N_{dst}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source ($N_{src}$)  Destination ($N_{dst}$)

1. Begin Migration
2. Iterative Copying
3. Atomic Handover

Steady State
Zephyr

[Elmore et al. SIGMOD 2011]

Focus

- Live migration in a shared nothing transactional DB (H2)
- No heavy-weight synchronization protocols or replication.
- No downtime, some aborted transactions.

Key Contributions

- First live migration for shared nothing DBMS.
- Minimal strain on source (scale up)
Init Mode

Freeze index wireframe and migrate

Owned Pages

Active transactions

Source

Destination

Un-owned Pages

Page owned by Node

Page not owned by Node
Dual Mode

Requests for un-owned pages can block

Old, still active transactions

Source

$T_{S_k,\ldots, T_{S_l}}$

$P_1$

$P_2$

$P_3$

$P_n$

$P_3$ pulled from source

Index wireframes remain frozen

Destination

$T_{D_1,\ldots, T_{D_m}}$

$P_1$

$P_2$

$P_3$

$P_n$

Page owned by Node

Page not owned by Node

New transactions
Finish Mode

Pages can be pulled by the destination, if needed

Completed

Source

\[ P_1, P_2, \ldots, P_n \]

Destination

\[ T_{D_{m+1}}, \ldots, T_{D_n} \]

\[ P_1, P_2, \ldots \text{ pushed from source} \]

Page owned by Node

Page not owned by Node
“Cut Me Some Slack”: Latency-Aware Live Migration for Databases

[Barker et al. EDBT 2012]

Focus

Interference aware live migration

Key Contributions

Throttles migration to minimize impact

Implementation with no internal modification
Slacker Approach

Uses hot backup to migrate
   Snapshot, Recover, Delta Shipping, & Handover

Throttle using a linux pipe limiter & piping backup

Use a PID controller (feedback loop on latency)
ProRea – Live Database Migration for Multi-tenant RDBMS with Snapshot Isolation

[Schiller et al. EDBT 2013]

Focus

Overcome some Zephyr shortcomings

Key Contributions

A proactive and reactive live migration
ProRea - Approach

Instead of 2PL based on SI

Proactively migrates hot pages

Reduced aborts from Zephyr

Implemented in PostgreSQL
In Closing
Many Other Issues

Pricing

Replication

  Swapping instead of migration [SWAT @ EDBT 2013]

Security / Privacy

Admission Control / Query Scheduling
Future Challenges
Additional resource isolation controls
  Query processing, buffer management, etc
SLOs / SLAs
  Workload or resource based
  Multi-user (application, data scientist, developer, C-level)
Data sharing
Better workloads
Analytics
Thanks!