CSE 544
Principles of Database Management Systems

Alvin Cheung

Fall 2015

Lecture 17 – DBMS in the Real World

References

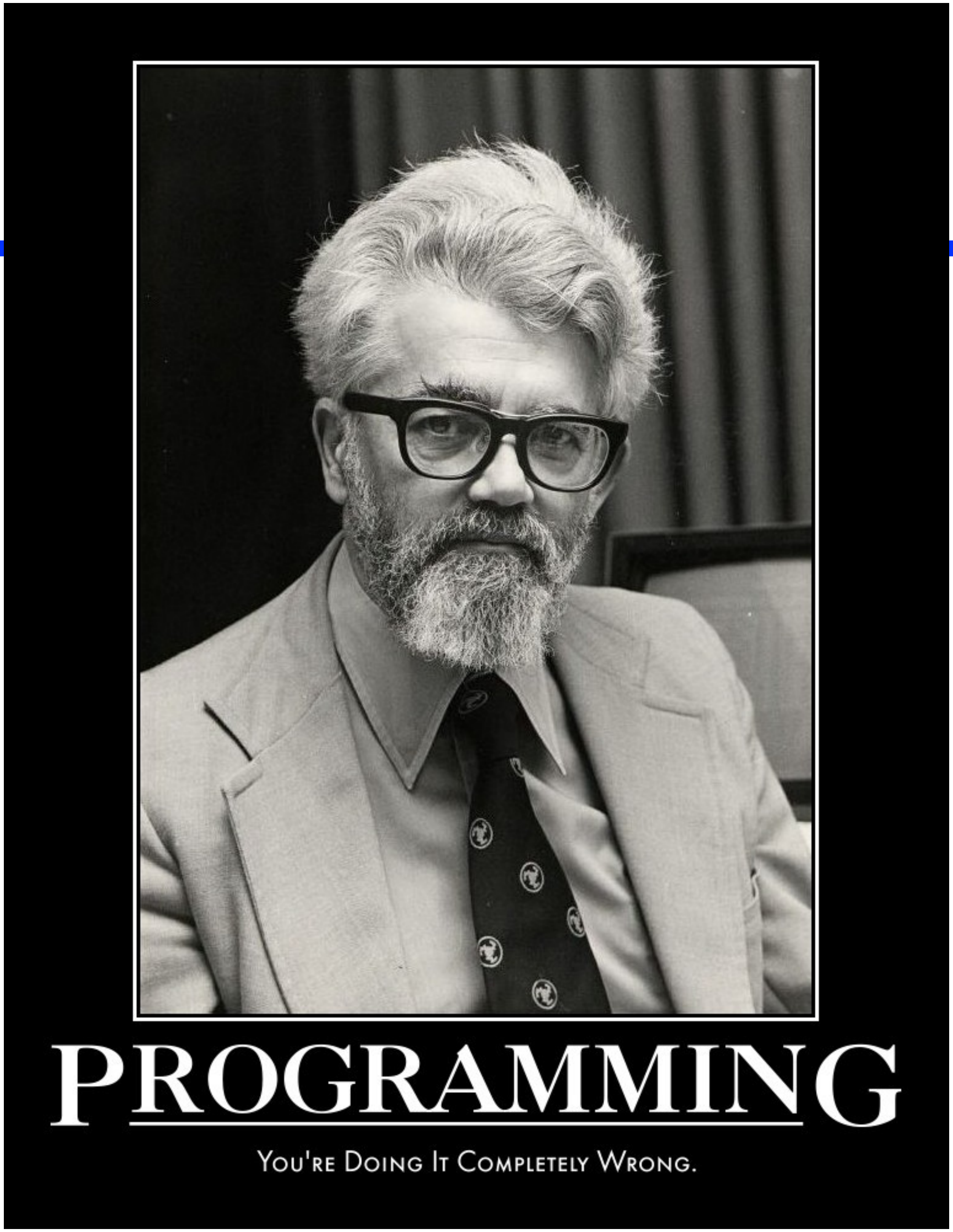
- Narasayya et al, SQLVM: Performance Isolation in Multi-Tenant Relational Database-as-a-Service. In CIDR 2013.
- Optional: Elmore et al, Characterizing tenant behavior for placement and crisis mitigation in multitenant DBMSs. In SIGMOD 2013.

Outline

- Cloud computing
- Multitenancy for Databases as a Service
- Writing database applications

Cloud Computing

- A definition
  - “Style of computing in which dynamically scalable and often virtualized resources are provided as a service over the Internet”
- Basic idea
  - Developer focuses on application logic
  - Infrastructure and data hosted by someone else in their “cloud”
  - Hence all operations tasks handled by cloud service provider
- Some history
  - "computation may someday be organized as a public utility” (John McCarthy – 1960)
  - 1996 Hotmail “Software as a Service”
  - 1999 Salesforce.com offers enterprise-class “Software as a Service”
  - 2006 Amazon Web Services with EC2
  - And now it’s commonly used


PROGRAMMING
You're Doing It Completely Wrong.

Service, Service, Service

- Infrastructure as a Service (IaaS)
  - Virtual machines, storage, and networking
  - Example: Amazon EC2
- Platform as a Service (PaaS)
  - Execution runtime, database, web server, development tools, …
  - Example: Google App Engine
- Software as a Service (SaaS)
  - Entire applications
  - Example: Google Docs
- Database as a Service (DaaS)
  - What this lecture is about
  - Example: EC2, Azure

Why DaaS?

- Running a DBMS is challenging
  - Need to hire a skilled database administrator (DBA)
  - Need to provision machines (hardware, software, configuration)
  - Problems:
    - If business picks up, may need to scale quickly
    - Workload varies over time
- Solution: Use a DBMS service
  - All machines are hosted in service provider's data centers
  - Data resides in those data centers
  - Pay-per-use policy
  - Elastic scalability
  - No administration!

Outline

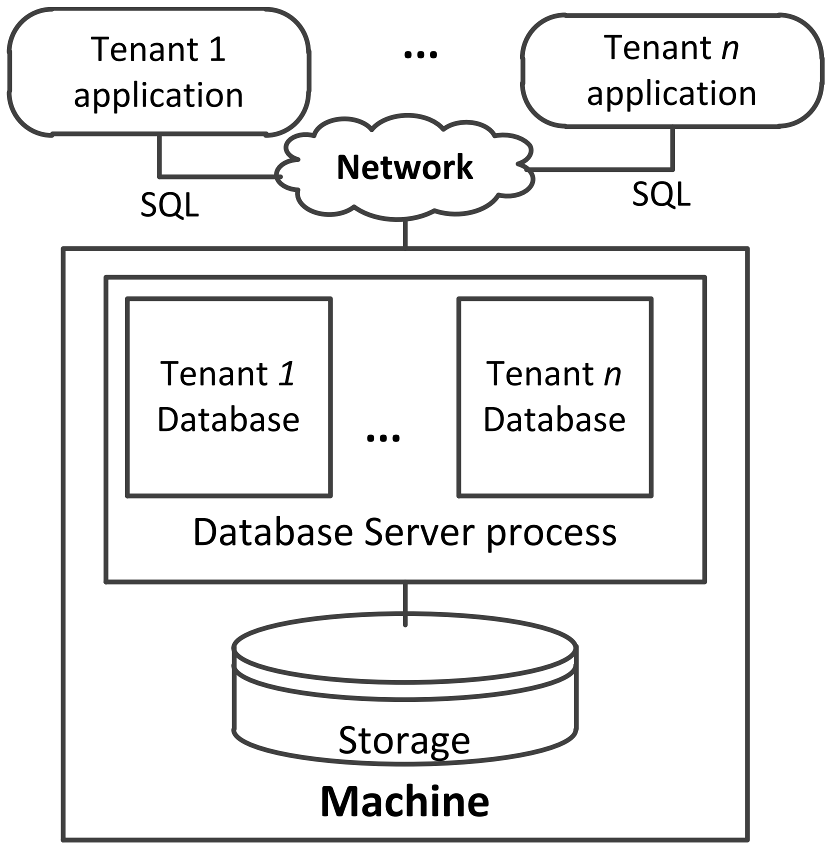
- Cloud computing
- Multitenancy for Databases as a Service
- Writing database applications

Multitenancy Problem

- Given a DBMS as a cloud service, how to support multiple tenants?

1. Each tenant runs in its own virtual machine(s)
   – For example Amazon AWS
2. Tenants share the same DBMS instances
   – For example SQL Azure
3. Tenants data is stored in a single table
   – For example force.com (underlying platform for salesforce.com)

Multitenancy Problem



Replication and Multitenancy

- Replication: same data store cloned multiple times across different nodes
  - For high availability and recovery

- Multitenancy: multiple, different data stores packed into the same node
  - For elasticity and DaaS

Tenant Placement

- Many tenants need less than the capacity of one machine
- How to consolidate many tenants on a few servers?
  - Also called "tenant packing"
- Question 1: Which tenants can be placed together?
  - Want to avoid interference
  - One challenge is that tenant workloads vary over time
- Question 2: How many tenants can we place together?
  - Trade-off between over-provisioning and over-booking

Tenant Migration

- When conditions change and SLAs are violated

- Need to move tenants
  - Which tenant to move?
  - How to perform the migration with minimum disruption?

Some Solutions

- Delphi: Self-managing controller for a multitenant DBMS

- Pythia: Learn behavior through observation
  - Tenant behavior
  - Node behavior
  - Uses database-level attributes
  - Assigns a class to each tenant and determines which tenant classes can be colocated
  - Assigns classes to packings: good, good with underutilized resources, or bad

Tenant Model

- DBMS-agnostic database-level performance measures
  - Write percent (insert, delete, updates)
  - Avg operation complexity: avg nb of pages accessed by tx
  - Percent cache hits
  - Buffer pool size: nb pages allocated to tenant
  - Database size
  - Throughput (transactions per second)
- Tenant labels
  - D: Disk IOPS, T: Throughout, and O: Operation complexity
  - Each resource type range is split into buckets
  - Tenant labels: DS-TS-OS

Node Model

- One feature per node: packing vector
  - One cell per tenant class
  - Value in cell is the number of tenants of that class
- Model learns mapping
  - From feature vector
  - To quality of packing: under, good, over
  - Apply model during runtime to schedule each tenant

Crisis Detection and Mitigation

- Periodically collect a snapshot of system state
- For each snapshot, classify tenants
  - Tenant class is aggregate class over time-window W
    - Example: $\{0.8c_j, 0.2c_k\}$
- If packing is bad, use hill-climbing to find a good packing
  - Consider all potential migrations of one tenant
  - Perform the move that yields the largest improvement
    - Naïve cost function minimizes the number of nodes labeled as "over"
      - Not good because algorithm tends to overload one node completely
    - Better cost function assigns a confidence to each node of being over
      - Consider only nodes with high confidence of being over
      - Minimize the weighted sum of tenants being on an overloaded node
  - Continue until cannot improve any more

A Case Study: SQLVM

- An abstraction to express performance characteristics in multi-tenant DBMS
- Resource scheduling is based on the given performance abstraction
- Metering capabilities to ensure each tenant is operating within resource bounds
- Implemented on Microsoft Azure platform

Performance Abstractions

- CPU
- I/O
- Memory
- Why were these chosen?

Abstracting the CPU

- T_i : slice of time on CPU core for tenant i
  - Actual amount of time dependent on metering interval, clock speeds, etc
  - Can also be defined as % of total available CPU cycles
  - Not pinned to any specific core in the system
    - Why?

- Metering:
  - Monitor job usage over a fixed period of time (metering interval)
  - Ensure that at least the guaranteed % of time has been allocated to tenant

- Enforce mechanism:
  - Job scheduler decides which tenant gets to use the CPU

Abstracting I/O

- Disk throughput (i.e., # of I/O operations per second)
- Disk bandwidth (i.e., # of bytes read / written per second)

- Reserve certain throughput / bandwidth to each tenant

- Metering:
  - Measure amount of I/O operations over each metering period
  - What if requests come in bursts?
  - Shared I/O?

- Enforce mechanism:
  - Disk controller determines which disk request to service

Abstracting Memory

- Buffer pool pages
- Working memory for each query operator

- Reserve certain amount of memory for each tenant
  - Each tenant thinks it actually holds that amount of memory to itself
  - Why do this?

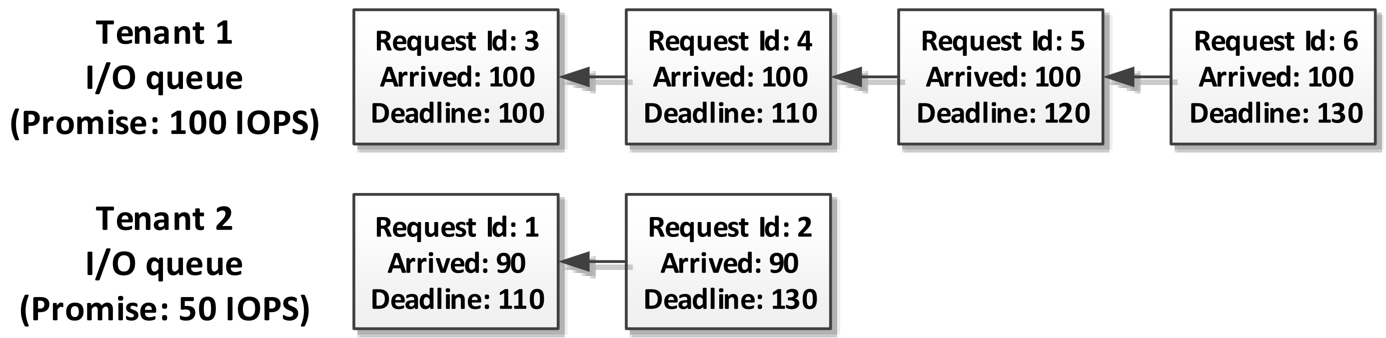
- Metering:
  - Measure number of memory pages each tenant holds

- Enforce mechanism:
  - Buffer page manager (recall HW2)

Scheduling

- Each query comes with requests for CPU, I/O, and memory
- Each tenant combines all requests and sends them to the underlying OS
  - OS then determines how to allocate physical resources
  - Implemented as a hypervisor (virtual machine) layer

Scheduling Example



Scheduling Algorithms

- First come first served
  - Up until promised limit
- Round-robin across tenants
- Priority-based
  - Based on Service Level Agreements (SLAs) with each tenant
- Machine learning based models

- Many other possibilities as discussed

Challenges

- How to ensure accurate accounting?

- What happens when a tenant violates its allocated budget?
  - Queries are still running on tenant's machine
  - Need to be careful when doing migration / eviction of tenants

- How does migration take place?
  - What needs to be moved?

Outline

- Cloud computing
- Multitenancy for Databases as a Service
- Writing database applications

Issuing Queries to DBMS

- Write SQL text on a command prompt provided by DBMS
  - These are called Command Line Interfaces (CLIs)
  - All major DBMS implementations provide this (HW3)

- Write queries graphically
  - Data stream systems (e.g., Aurora)
  - Essentially the same except that queries are constructed via GUIs
  - Advantages?

CLI

- This has been the only way to interact with DBMSs for the first 20 years or so
- Database applications = accounting, business processing
- Users were clerks / accountants in large corporations

IBM System/38

Rise of Programming Languages

- 3rd generation "high level" general purpose programming languages caught on starting in the 80s
- Users start to write applications in those languages instead
  - Procedural languages: Fortran, COBOL, C
  - Object-oriented languages: CLU, C++, Java
- Problem: those languages do not work well with SQL
  - Famous example: "impedance mismatch"

“Impedance” Mismatch

- Issues between general-purpose programming languages and query languages:
  - Data types
  - Object encapsulation, inheritance, polymorphism (for object oriented languages)
  - Transactions
  - Schema changes
  - Imperative and declarative programming styles
  - Security

Dealing with Impedance Mismatch

- Don't use a DBMS (!)

- Object-Oriented DBMS (OO-DBMS)
  - Object instances directly stored in DBMS
  - Write GP code to access objects directly (no more SQL)
  - (yet another data model)
  - Popular in the 90s

  - Very difficult to optimize
    - Pointers everywhere! (IMS?)

Database Drivers

- RDBMS start to provide **drivers** for applications to access persistent data
- Idea: applications embed SQL strings within GP code
- Examples with standardized interfaces:
  - ODBC (Open Database Connectivity)
    - Mainly for C/C++ programs
  - JDBC (Java Database Connectivity)
- Each DBMS provides its own driver implementation

Using Database Drivers

```
Connection conn = null;
Statement stmt = null;
Class.forName("com.mysql.jdbc.Driver");
Connection conn = DriverManager.getConnection(DBMS_NAME, username, password);
Statement stmt = conn.createStatement();
String sql = "SELECT id, first, last, age FROM Employees";
ResultSet rs = stmt.executeQuery(sql);

while(rs.next()) {
  int age = rs.getInt("age");
  String name = rs.getString("name");
  ... ...
}
rs.close();
stmt.close();
conn.close();
```

Issues with Drivers

- Users need to learn two languages
- Every driver is slightly different in its calling syntax
- Type safety?
- Software engineering nightmare
- Inefficient data serialization between DBMS and application
  - But at least you don't need to write the serialization code

Rise of the Internet

- Web applications become popular in the 2000s
- Database applications = web applications
  - online forums, online stores, etc
- Easy integration with the web server is important

Web Applications

- Typical three-tier web applications
  - Frontend (browser, phone, etc)
  - Middle tier (web server hosting the application)
  - Backend (databases)

- Embedding SQL strings within application becomes tedious and clumsy
  - You only need to learn SQL, php, Javascript, HTML, … to write web apps

Web Frameworks

- MVC design pattern
  - Model
    - Database schemas (e.g., SQL)
  - View
    - Presentation layer (e.g., HTML)
  - Controller
    - Application logic (e.g., php)

- Compare this to ER diagrams

Web Frameworks

- Idea:
  - Declare models up front
    - i.e., what need to be persistently stored
  - Implement application logic using general purpose language
  - Web framework generates all necessary SQL and create database tables, indexes, etc

- Issue: still need to learn another language for the presentation layer
  - Some frameworks provide that capability as well

Web Frameworks

ASP.NET	PHP Fat-Free Framework	Koa	Zend	Stripes
AngularJS	Lift	web2py	Google Web Toolkit	Grok
Ruby on Rails	CherryPy	(fab)	Play	Zope
ASP.NET MVC	Restlet	Gin	Yii	Orbit
Django	Lithium	Vaadin	Sails.js	TurboGears
Laravel	OpenUI5	Yesod	Sinatra	Merb
Meteor	Tapestry	Compojure	Grails	Ramaze
Spring	Flight	Revel	Tornado	Ratpack
Express	CompoundJS	Martini	Phalcon	Aura
CodeIgniter	ZK	Mithril	Dojo	seaside
Symfony	Flatiron	beego	Struts	Zotonic
Ember.js	Noir	Ring	web.py	PureMVC
Flask	Catalyst	SproutCore	Bottle	Tipfy
JSF	Nitrogen	Mojolicious	Pyramid	Horde
CakePHP	Snap	SilverStripe Sapphire	Kohana	Cappuccino
Flex	Camping	Scalatra	Wicket	Swiz

Model Code Example

```
from django.db import models

class Question(models.Model):
  question_text = models.CharField(max_length=200)
  pub_date = models.DateTimeField('date published')

class Choice(models.Model):
  question = models.ForeignKey(Question,
                               on_delete=models.CASCADE)
  choice_text = models.CharField(max_length=200)
  votes = models.IntegerField(default=0)
```

Retrieving Objects

```
from polls.models import Question, Choice

Question.objects.all()
q = Question(question_text="What's new?",
             pub_date=timezone.now())
q.save()

q.id
>> 1  # automatically assigned by the DBMS
```

Issues with Web Frameworks

- How are objects stored?
  - Physical design problem
- How to debug?
- What if object layout needs to be changed?
- Generated queries are inefficient
  - The "N+1" problem

Recall: BCNF Decomposition

$R(A_1, ..., A_n, B_1, ..., B_m, C_1, ..., C_p)$

$R_1(A_1, ..., A_n, B_1, ..., B_m)$

$R_2(A_1, ..., A_n, C_1, ..., C_p)$

R_1 = projection of R on $A_1, ..., A_n, B_1, ..., B_m$
R_2 = projection of R on $A_1, ..., A_n, C_1, ..., C_p$

Theorem If $A_1, ..., A_n \rightarrow B_1, ..., B_m$
Then the decomposition is lossless

Note: don't necessarily need $A_1, ..., A_n \rightarrow C_1, ..., C_p$

Example

Patient

pno	name	zip
1	p1	98125
2	p2	98112
3	p1	98143

PatientOf

pno	dno	since
1	2	2000
1	3	2003
2	1	2002
3	1	1985

How to reconstruct a Patient object?

ORM: Use nested selects!

Integrating Queries into Languages

- Make query constructs first-class citizens in the programming language itself
- Examples: Microsoft LINQ

```
var numbers = DB.Tables["Numbers"].AsEnumerable();
var numsPlusOne = numbers.Select(n => n.Field<int>(0) + 1);
foreach (var i in numsPlusOne) {
    Log.WriteLine(i);
}
```

- Code is compiled by the C# compiler, which understands query operations

Conclusion

- DaaS is becoming increasingly popular
  - AWS, Azure, Google, and many other cloud service providers
- Multitenancy is an active area of research
  - Modeling
  - Migration
- Various ways to write DB applications
  - CLI
  - Drivers
  - Frameworks
  - New languages