CS263: Runtime Systems
Winter 2016

http://www.cs.ucsb.edu/~cs263
Sign up for piazza now please (the link’s on this page)

Prof. Chandra Krintz
Laboratory for Research on Adaptive Computing Environments (RACELab)
Computer Science Department
Harold Frank Hall (HFH) 2153
ckrintz@gmail.com
SUMMARY FROM LAST TIME

- Dennard scaling is ending
  - Demand for performance is not satiated
- Widespread belief (hope?) that parallelism will save the day
- Too little attention to improving efficiency across the hardware-software stack
- Need sustained focus on improving software performance
  - New tools, techniques, and hardware
  - More attention to developing efficient systems, not individual components
CS263 GOALS

- Learn and understand the various core functions of modern runtime systems
- Gain hands-on experience with a modern distributed runtime system (cloud platform)
  - Understand the web-based software architecture
- Dig deeper into the software stack that ultimately executes the program
  - Managed runtime system, aka (programming language) virtual machine
  - We’ll focus on Java (but I’ll point out the diffs with other languages as appropriate)
- Programmer productivity aids that impact performance (and how we try to recover it)
  - Portability of apps
  - Garbage collection
CS263 Course Topics

- Software, services, and their interoperation
  - Google App Engine – modern cloud app platforms
- Modern languages and their managed runtime systems
  - OO review
  - Implementations
  - Java focus
- Garbage collection
- Execution: interpretation, dynamic/JIT compilation
- Adaptive (profile-driven) optimization
External trends and influences have changed language/runtime support -- Your project in this class takes advantage of each

- Hardware/architecture evolution
  - Low cost, HPC, memory-rich, multicore, virtualization
  - Moore’s law (Dennard scaling is dead)

- Apps built by composition of components
  - Components increasingly becoming web-accessible services
  - APIs: gateway to digital assets, business opportunities

- ...
**Changes in Software Development**

- Each component in its own managed runtime
  - Fault isolation
  - Separate concerns
  - Can execute anywhere
    - **Same/different machine**
    - Optimize/specialize runtime

- Emphasis moves to
  - Locating/naming
  - Coordination
    - Data transfer and persistence
  - Communication
    - Interprocess communication (IPC)

\[ \text{JVM=VM=MRE=Runtime} \]
\[ = \text{execution engine} \]
External trends and influences have changed language/runtime support -- Your project in this class takes advantage of each

- Hardware/architecture evolution
  - Low cost, HPC, memory-rich, multicore, virtualization
  - Moore’s law (Dennard scaling is dead)
- Apps built by composition of components
  - Components increasingly becoming web-accessible services
  - APIs: gateway to digital assets, business opportunities
- Distributed computing
  - Remote execution, multi-tasking, containerization, cloud
- The people who are developing applications/software
  - Productivity programmers vs specialists/experts
- Cloud platforms have emerged to simplify process
  - Automatically configure, deploy, manage apps, APIs, and services
External trends and influences have changed language/runtime support -- Your project in this class takes advantage of each

- Hardware/architecture evolution
- Apps built by composition of components
  - Web services
- Distributed computing
- Programmer productivity
  - High-level languages, garbage collection
- **Cloud platforms** bring it all together
CS263 PROJECT

- 50% of class grade (1-2 person groups)
  - Java Google App Engine App (free tier)
    - Required APIs: datastore (no JPO/JDA,Slim3), task queue, and memcache (no Jcache)
    - Must expose a **GET and POST REST API for each GAE API/service**
      - REST APIs must use JSON to send/receive data
      - Turnin includes **curl commands** that work on Linux to read (GET) and write (POST) each of the services
    - Use of **JAX-RS**
      - Use of **maven** to package and build the application
      - Selenium or other tool script that exercises app APIs, with usage in project README
  - Public github repo, fully documented, **commit (100+ lines/wk) required over time incl. tests**
  - 15-20 minute public, recorded (youtube, vimeo, ...) demo and presentation
    - Overviewing project, contributions, novelty, experience, performance evaluation, and demonstrating app execution
  - **Turnin**: link to github repo, link to running app, link to video, file with tested curl commands
  - App to be kept running until graded (Friday of finals week)

All Deadline Dates
Posted on Website Schedule
CS263 EVALUATION

http://www.cs.ucsb.edu/~cs263/

- 10% Class attendance and participation
  - Read papers, come prepared to ask/answer questions on the topic
  - Assignments (on schedule) in and out of class
- 40% Midterm
  - Review in class prior, midterm in class (closed notes/books/etc)
- 50% Project (1-2 person groups)
  - Weekly code commits (starting with Assignment 3)
    - Public github repo
  - Youtube presentation/demo at end
JAX-RS

- A standard Java API for developing RESTful services
- Implement your service in plain old Java and add some annotations to expose it as a RESTful service
- Supported by a variety of programming frameworks and runtimes
  - Jersey, Apache CXF, Apache Wink, IBM WebSphere, Oracle WebLogic, RESTEasy, RESTlet etc
- Links under today’s schedule (required reading & other resources)
  - REST, JAX-RS, maven, App Engine
CURL

- Powerful command-line tool for testing HTTP endpoints

- **GET:**
  - `curl -v http://test.com`

- **POST (Form):**
  - `curl -v -X POST -d "key=value" http://test.com`

- **POST (XML):**
  - `curl -v -X POST -d @test.xml -H "Content-type: application/xml" http://test.com`
ASSIGNMENTS 1, 2, AND 3

- **Asgn1: in class next Tuesday**
  - Setup App Engine and a simple app deployment locally (for testing: dev app server) and on Google’s resources

- **Asgn2: in class next Thursday**
  - Deploy a more complicated application that uses the Datastore (guestbook)

- **Asgn3: out of class (Due Thurs Jan 21 by start of class)**
  - No class Jan 19 (work on this assignment): Add services, REST support
  - Also part of this assignment: github, vision statement, etc. for your project
    - See web page for details

- **Lays the groundwork for project – extend with your innovation**
  - App Engine, key cloud services, REST/JAX-RS, curl, debugging/deployment

*Bring your laptops to class all next week*
An Introduction to Google App Engine Platform as-a-Service (PaaS)

Technologies
Chandra Krintz, UCSB
Your IT, Devs, DevOps

Managed by:
• Vendor software (public) e.g. AWS
• IaaS software (on-premise)

Security & Integration

Your Devs

Managed by:
• Vendor software (public) e.g. Google App Engine, Azure
• PaaS software (on-premise)
Google’s platform as-a-service – one of the **first** public PaaS’s
- Hosting service for web apps, services, and mobile backends
- No notion of “server”
- Google’s platform as-a-service – one of the **first** public PaaS’s
  - Hosting service for web apps, services, and mobile backends
- The result of over a decade of studying Googlers
  - Make them more productive, innovative, and satisfied
- Google’s platform as-a-service – one of the **first** public PaaS’s
  - Hosting service for web apps, services, and mobile backends
- The result of over a decade of studying Googlers
  - Preview release in 2008, GA in 2011
  - Now over **4 million** active apps, ½ of all Internet accesses/week use it
- Decouples app/innovation from common services
  - Share scalable services across apps
  - Automatically manages and scales apps + service ecosystem
- Decouples app/innovation from common services
  - Share scalable services across apps
  - Automatically manages and scales apps + service ecosystem

**App Engine**

**APIs**

- Data Storage & Processing (NoSql, SQL,...)
- Security & Authentication
- Monitoring & Logging
- Messaging & Communications
- User Credentialing
- Web Hosting & Serving

Programming Languages:
- python
- java
- The Go Programming Language
- php
- Everything is a web request or background task
- Sandbox execution
  - Restrictions for scale/safety, quotas on free use
    - No file system access
    - Processing limits (frontends and tasks)
    - Data persistence via Datastore, memcache, and Cloud SQL
    - Language libraries limited to whitelist
- Quotas (free and billed)
  - In/Out bandwidth
  - Datastore usage
  - Other APIs (Mail, messaging, URL Fetch, …)
Traditional Website Implementation

Reverse Proxy

Apache

MySQL

MySQL

MySQL

MySQL

MySQL

MySQL
- App Master: orchestrates everything
- AE Front End: provides load balancing on App Servers and Static Servers
- App Servers provide runtime environment for Application Instances
- Frontend instance (Application instances)
  - Computing units
    - Not an App Engine Front End
  - Automatically scaled (parameters controlled by you with defaults)
  - Dynamically created and deleted = low cost
  - Enforce fast response and stateless design
  - Suitable for processing short-lived requests
    - Deadlines: HTTP requests: 32MB, 60s; 10min background tasks
  - Different sizes (memory/CPU): F1,F2,F4,F4_1G
    - Default: F1 – 128MB, 600MHz, $0.05/hr; F4_1G – 1GB, 2.4GHz, $0.30/hr
- Billed on instance hours: minute time granularity (+15mins startup)
- Backend instance (also computing units)
  - Statically created and deleted = higher cost
    - Configurable memory, CPU, elasticity
  - No limit for response time, supports stateful design
  - Suitable for batch processing, run indefinitely (attempted)
- Billed similarly to frontend instances

- Quotas: free, billable, safety limits
  - Daily, refreshed at midnight Pacific time
  - Per minute quotas, resource requests are delayed/denied
  - Creates, API calls/operations, data transmitted/size, storage, indexes, files uploaded, logging, sockets, HTTPs
- **Datastore** – key/value object persistent storage
  - Fast, replicated, and scalable for large-scale data
- **Memcache** – key/value in-memory cache (not persistent)
- **Task queue**, cron, pipelines, map reduce
- **Blobstore**: key/value persistent storage for large objects
- Users, mail, search, XMPP, URL fetch

***Bold APIs are required to be included in your project/app.***
Datastore – key/value object persistent storage
- Fast, replicated, and scalable for large-scale data
- Easily extensible entity structures (kinds), less than 1MB in size
- Simple API: put, get, delete, range_query (SQL subset)
  - Only simple queries supported, limited indexing support, no shared counters
- Reads are cheap, writes are expensive,
  - Writes: parallelize and optimize low contention
    - Trade-off space for low contention, avoid sharing
- Strongly consistent entity writes (row-level atomic updates) – but eventually consistent across servers
  - Limited transaction support for multi-entity atomic updates

See also: http://highscalability.com/numbers-everyone-should-know
Kind, Entity and Key

**Key**

BlogEntry
- Key: 1234
- name: joe@ex.com
- message: xxxxx
- date: 1/1/2012 12:32

User
- Key: joe@ex.com
- email: joe@ex.com
- followees: [usr2@ex.com, usr3@ex.com]
- followers: []

User
- Key: usr2@ex.com
- email: usr2@ex.com
- followees: []
- followers: [joe@ex.com]

**Properties**

**Kinds**

**Entities**
Creating an Entity with Java Low-Level API

```java
DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();
String name = "Joe";
Key empKey = KeyFactory.createKey("Employee", name);
Entity employee = new Entity(empKey);
employee.setProperty("name", name);
employee.setProperty("hireDate", new Date());
employee.setProperty("attendedTraining", true);
datastore.put(employee);
```

With Transactions

```java
DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();
Transaction txn = datastore.beginTransaction();
try{
    Key empKey = KeyFactory.createKey("Employee","Joe");
    Entity emp = datastore.get(empKey);
    //read/write emp object
    Address addr = Address(parent=empKey);
    addr.setProperty(…);
    datastore.put(emp);
datastore.put(addr);
    txn.commit();
} finally {
    if (txn.isActive()) { txn.rollback(); }
}
```
Datastore Code Example

Creating an Entity with Java Low-Level API

```java
DatastoreService datastore =
    DatastoreServiceFactory.getDatastoreService();
String name = "Joe";
Key empKey =
    KeyFactory.createKey("Employee", name);
Entity employee = new Entity(empKey);
employee.setProperty("name", name);
employee.setProperty("hireDate", new Date());
employee.setProperty("attendedTraining", true);
datastore.put(employee);
```

Low-level API -- what you’ll use for your project

- **Best performance, more coding**
- JDO/JPA
- More portability; Java standard APIs
- Third-party frameworks
  - Objectify, Twig, Slim3, …
  - Sophisticated features with good performance

With Transactions

```java
DatastoreService datastore =
    DatastoreServiceFactory.getDatastoreService();
String name = "Joe";
Key empKey =
    KeyFactory.createKey("Employee", name);
Entity employee = new Entity(empKey);
employee.setProperty("name", name);
employee.setProperty("hireDate", new Date());
employee.setProperty("attendedTraining", true);
datastore.put(employee);
```

```java
Transaction txn = datastore.beginTransaction();
try{
    Key empKey =
        KeyFactory.createKey("Employee","Joe");
    Entity emp = datastore.get(empKey);
    //read/write emp object
    Address addr = Address(parent=empKey);
    addr.setProperty(...);
    datastore.put(emp);
    datastore.put(addr);
    txn.commit();
} finally {
    if (txn.isActive()) { txn.rollback(); }
}
```
Query is Executed as Index Scan

```
SELECT * FROM Person
WHERE height < 72
ORDER BY height DESC
```

Datastore Query

Index table for height

| height: 75 |
| height: 73 |
| height: 71 |
| height: 70 |
| height: 68 |
| height: 67 |
| height: 64 |

Entities in the query result

- first_name: John
  height: 71
- first_name: Bob
  height: 70
- first_name: Kate
  height: 68

Range Scan on Bigtable

Google Cloud Platform
- No joins, aggregate functions, search
  - No notion of total or partial counts
- Sorts are performed ahead of time via index construction
- Single property
- Composite (multi-property)
- Entity groups – Restricted: 1 update per second
  - Defines the scope of a transaction
  - Hierarchical relationships: Parent->Child->GrandChild
    - ACID transactions within entity groups (optimistic locking)
    - Ancestor queries
- All properties have single indexes auto-generated (both asc/desc)
- All composite indexes must be specified explicitly
- Note that Datastore data size (bytes, quota, billed)
  - Includes all of its indexes
- Indexes in datastore-indexes-auto.xml
  - appName/target/appName-1.0-SNAPSHOT/WEB-INF/appengine-generated/datastore-indexes-auto.xml
  - Development server automatically adds suggestions by running app
  - An index is defined on a list of properties of a given kind (entity type) with a corresponding order (asc/desc)
    - For use with ancestor queries, the index may also optionally include an entity's ancestors
- All properties have single indexes auto-generated (both asc/desc)
- All composite indexes must be specified explicitly
- Note that Datastore data size (bytes, quota, billed)
  - Includes all of its indexes

- Indexes in datastore-indexes-auto.xml

```xml
<datastore-indexes>
  <datastore-index kind="Greeting" ancestor="true" source="manual">
    <property name="date" direction="desc"/>
  </datastore-index>
</datastore-indexes>
```
- **Datastore** – key/value object persistent storage
  - Fast, replicated, and scalable for large-scale data

- **Memcache** – key/value in-memory cache (not persistent)
- **Task queue**, cron, pipelines, map reduce
- **Blobstore**: key/value persistent storage for large objects
- Users, mail, search, XMPP, URL fetch

***Bold APIs are required to be included in your project/app.***
- Task: unit of work
  - Write object to datastore
  - Send an email
- All versions of an application share queues
- Push queue for auto execution
- Pull queues to programmatically consume tasks
- Task have unique names
  - Generated automatically if not assigned
  - Insert new task with same name will fail
• A simple way to perform work in your app, outside of a user request

Features:
• Executed ASAP
  May cause new instances
  Frontend or Backend
  - 10min or unlimited
  - Max 100K task size

Features:
• Task leased by worker
  REST interface with ACL
  - Can be outside App Engine
  - Max 1MB task size
Push Queues

- Queues are named and configurable (queue.xml/queue.yaml)
  - Processing rate (e.g. 60/m == 1/s)
  - Token bucket size (default = 5)
  - Max concurrent requests (default = unlimited)
  - Configurations can be overridden for individual tasks

Unless specified otherwise, tasks are executed on the version of the app which spawned it
<!-- index.html file served from "/" -->
<html>
<body>
<p>Enqueue a value for worker.</p>
<form action="/enqueue" method="post">
<input type="text" name="key">
<input type="submit">
</form>
</body>
</html>

// The Enqueue servlet is mapped to "/enqueue"
import com...taskqueue.Queue;
import com...taskqueue.QueueFactory;
import static com...taskqueue.TaskOptions.Builder.*;

public class Enqueue extends HttpServlet {
    protected void doPost(HttpServletRequest request,
            HttpServletResponse response) 
        throws ServletException, IOException {
        String key = request.getParameter("key");

        // Add the task to the default queue.
        Queue queue = QueueFactory.getDefaultQueue();
        queue.add(withUrl("/worker").param("key", key));
        response.sendRedirect("/");
    }
}

// The Worker servlet mapped to the "/worker"
public class Worker extends HttpServlet {
    protected void doPost(HttpServletRequest req,
            HttpServletResponse res) 
        throws ServletException, IOException {
        String key = req.getParameter("key");

        // Do something with key.
    }
}
Add task
   Queue q = QueueFactory.getQueue("pull-queue");

Lease (here, 100 tasks) then delete
   tasks = q.leaseTasks(3600, TimeUnit.SECONDS,100);
   //do the work
   q.deleteTask(tasks);
- Download the SDK from Google
- Create an app ID via the admin console: https://appengine.google.com/
- Program your app
  - Including **program configuration files**
  - Autogenerated with GoogleAppEngineLauncher (Mac, Windows)
- app.yaml – in top level app directory
  - # start single line comments
  - POSIX regex syntax

- Autogenerated by development server
  - WEB_INF/appengine-web.yaml
  - Autogenerates WAR xml files
  - Indexes autogenerated: WEB_INF/index.yaml

- application: myapp
  version: 1
  runtime: java
  api_version: 1
  threadsafe: true

  handlers:
  - url: /
    script: home.app

  - url: /index\..html
    script: home.app

  - url: /stylesheets
    static_dir: stylesheets

  - url: /(.*\.(gif|png|jpg))
    static_files: static/\1
    upload: static/(.*\.(gif|png|jpg))

  - url: /admin/.*
    script: admin.app
    login: admin

  - url: /.*
    script: not_found.app

- https://developers.google.com/appengine/docs/java/config/appconfig
- https://cloud.google.com/appengine/docs/java/configyaml/appconfig_yaml
- Program your app, test with the SDK, generate Datastore indexes
- Update configuration files (fine tune as needed)
- Upload your app to Google’s resources for execution
- Command line or via Eclipse (we’ll use the command line)
- https://console.developers.google.com/project
- Top left menu icon:
- Usage
- Debugging help
- Datastore viewer!
  - Delete entities here
  - /_ah/admin in local dev server
- App Engine (PaaS)
- Compute Engine (IaaS)
- Cloud Storage
- Cloud SQL
- BigQuery

- Technology APIs…

- Linked via service accounts
  - Billing must be enabled
  - https://developers.google.com/accounts/docs/OAuth2#serviceaccount
- Read EVERYTHING I’ve written for you in the tutorials
- Deploy to dev server (local test execution engine)  
  http://localhost:8080
  `mvn appengine:devserver`  //rebuilds and deploys
  `/_ah/admin`  //local datastore direct access
- Clean build and deploy – also deletes your local database
  `mvn clean install`
  `mvn appengine:devserver`
- Deploy to app engine (Google) only after local deploy works
  `mvn appengine:update`  https://appID.appspot.com
  Debug via Admin console; datastore data: Admin Console->Datastore->Entities
  https://console.developers.google.com/project