TCSS 562: SOFTWARE ENGINEERING FOR CLOUD COMPUTING

Cloud Computing:
Term Project

Wes J. Lloyd
School of Engineering and Technology
University of Washington - Tacoma

FUNCTIONAL VS. NON-FUNCTIONAL ATTRIBUTES OF SYSTEMS

- **Functional requirement:**
- Pertains to a system supporting a specific function
- What a system is supposed to do
- Testable with unit tests, integration tests, etc.

- **Non-functional requirement:**
- Specifies criteria used to judge how a system operates
- How a system should be (or behave)
- Considered as "quality" attributes of systems
- Testable by applying metrics to characterize degree of possessing a given quality

NON-FUNCTIONAL REQUIREMENT: HIGH AVAILABILITY

- ***The system should be highly available.***
- The system should be **99.9%** available per month
  - Maximum downtime: **43m 49.7s** monthly, **8hr 45min 36s** yearly
  - Functional attribute: system should notify users if there is an issue affects the availability or may cause downtime.
- Availability equation:

$$\text{AVAILABILITY} = \frac{\text{MTBF}}{\text{MTBF}+\text{MTTR}}$$

- MTBF: Mean time between failures
- MTTR: Mean time to Repair
- MTBF = ~1 month = 43,757 min; MTTR = 43 min
- AVAILABILITY = 43757 / 43800 = 99.9018265%

STRATEGIES FOR HIGH AVAILABILITY

- Replicate system resources in multiple data centers or cloud computing regions
- Use redundant infrastructure components
  - For load balancing, fault tolerance
- Report availability status via portal
- Allow users to immediately report outages
- Notification systems to alert system admins when system experiences an outage

- **Tradeoffs:**
- Highly available cloud resources are more expensive
- Replicating app components (e.g. database) adds cost

QUANTIFYING NON-FUNCTIONAL QUALITY ATTRIBUTES

- What are the "best" metrics to quantify non-functional quality attributes?
  - Consider ease/effort/time/cost of assessment
  - Relationship to expert opinion (e.g. correlation)
  - Relationship to other measures

FEEDBACK FROM 10/8

- Cohesion in Object Oriented Design

TYPES OF MODULARITY

- ***Soft modularity:*** TRADITIONAL
- Divide a program into modules (classes) that call each other and communicate with shared-memory
- A procedure calling convention is used (or method invocation)
- Object-oriented programming classic best practices:
- **Minimize coupling** between classes (OO) and modules
- **Maximize cohesion** between functions in classes (OO) and modules
  - Best practices lead to improved software reusability, maintainability, portability

TYPES OF MODULARITY - 2

- ***Enforced modularity:*** CLOUD COMPUTING
- Program is divided into modules that communicate only through message passing
- The ubiquitous ***client-server*** paradigm
- Clients and servers are independent decoupled modules
- System is more robust if servers are stateless
- May be scaled and deployed separately
- May also FAIL separately!

COUPLING AND COHESION

- **Object-oriented coupling**
- Degree of interdependence between software modules
- A measure of how connected two classes or modules are
- Captures the degree of the relationships between modules
- Coupling is usually contrasted with cohesion
- Low coupling often correlates with high cohesion
- High coupling often correlates with low cohesion

- **Object-oriented cohesion**
- Degree to which elements inside a class or module belong together
- Do the methods and data inside of a class interoperate with each other (***High cohesion***)? Or is the class a catch all bin of random functions (***Low cohesion***)?
  - E.g. "Util" class where random helper routines land… (***low cohesion***)

FEEDBACK - 2

- **Serverless vs. Classic Cloud –**
- *Is there a significant time difference between using one or the other?*

- **Topics related to the term project -**
- Service Composition

- Switchboard architecture

TCSS 562 TERM PROJECT

- Serverless cloud native application
- Choose one area to study:

- **Service composition**

- **Switchboard architecture**
  - Address COLD Starts
  - Infrastructure Freeze/Thaw cycle of AWS Lambda (FaaS)

- **Application flow control**

- **Data provisioning**

EXTRACT TRANSFORM LOAD DATA PIPELINE

- Service 1: **TRANSFORM**

- Read CSV file, perform some transformations
- Write out new CSV file

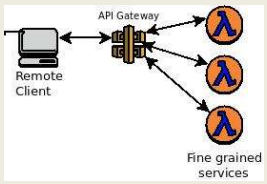
- Service 2: **LOAD**

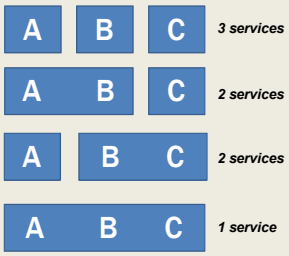
- Read CSV file, load data into relational database
- Cloud DB (AWS Aurora), or local DB (Derby/SQLite)
  - Derby DB and/or SQLite code examples to be provided in Java

EXTRACT TRANSFORM LOAD DATA PIPELINE 2

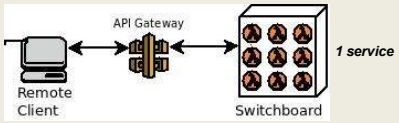
- Service 3: **EXTRACT**
- Using relational database, apply filter(s) and/or functions to aggregate data to produce sums, totals, averages
- Output aggregations as JSON

SERVICE COMPOSITION





SWITCH-BOARD ARCHITECTURE



Single deployment package with consolidated codebase (Java: one JAR file)

Entry method contains "switchboard" logic
Case statement that route calls to proper service

Routing is based on data payload
Check if specific parameters exist, route call accordingly

Goal: reduce # of COLD starts to improve performance

APPLICATION FLOW CONTROL

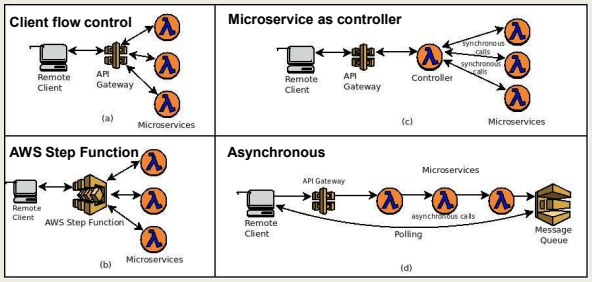
- **Serverless Computing:**
- AWS Lambda (FAAS: Function-as-a-Service)
- Provides HTTP/REST like web services
- Client/Server paradigm

- **Synchronous web service:**
- Client calls service
- Client blocks (freezes) and waits for server to complete call
- Connection is maintained in the "OPEN" state
- Problematic if service runtime is long!
  - Connections are notoriously dropped
  - System timeouts reached
- Client can't do anything while waiting unless using threads

APPLICATION FLOW CONTROL - 2

- **Asynchronous web service**
- Client calls service
- Server responds to client with OK message
- Client closes connection
- Server performs the work associated with the service
- Server posts service result in an external data store
  - AWS: S3, SQS (queueing service), SNS (notification service)

APPLICATION FLOW CONTROL - 3



DATA PROVISIONING

- Consider performance and cost implications of the data-tier design for the serverless application
- Use different tools as the relational datastore to support service #2 (LOAD) and service #3 (EXTRACT)

- **SQL / Relational:**
- Amazon Aurora (serverless cloud DB), Amazon RDS (cloud DB), DB on a VM (MySQL), DB inside Lambda function (SQLite, Derby)

- **NO SQL / Key/Value Store:**
- Dynamo DB, MongoDB, S3

QUESTIONS