CLOUD COMPUTING

UNIT-1

Cloud Programming Models

feedback/corrections: vibha@pesu.pes.edu

VIBHA MASTI

Cloud computing

- Computing at the internet scale, on demand, within minutes
- Keywords: ubiquitous, on-demand access to shared pool of configurable computing resources
- Data centers: house physical compute, storage, networking

Features of Cloud Computing

1. On-demand self-service
2. Broad network access
3. Resource pooling
4. Scalability
5. Rapid elasticity
6. Measured service

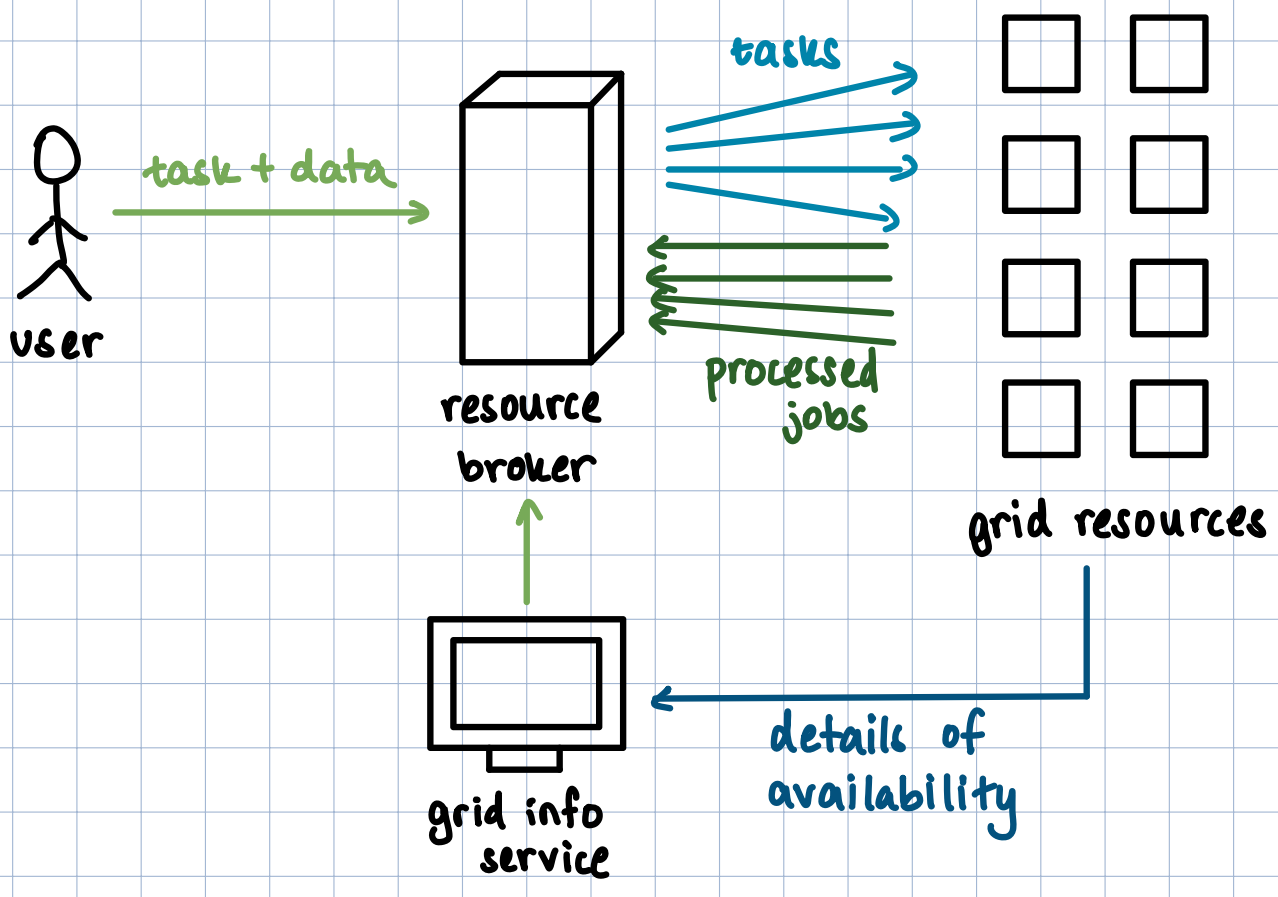
Benefits

1. Agility
2. Elasticity
3. Cost savings
4. Deploy globally in minutes

Evolution of CC

- **High performance computing (HPC)** - perform at high speeds for short periods
- **High throughput computing (HTC)** - handle large amounts of compute for longer periods
- **Distributed computing** — multiple autonomous computers communicating via message passing
- **Clusters** - connected computers that work together as a single system
  * each node performs same task
  * run diff instances of same OS
  * similar hardware
  * VPN
- **Parallel computing** — single system with multiple processors
  * centralized memory
  * distributed memory
  * degrees of parallelism
    → bit-level parallelism: increase word size
    → instruction-level: eg: parallel loop for adding 2 arrays
    → task level: sum of array using threads
    → data: threads to calc mean, std dev in parallel
  * parallel computing architecture
    → multi-core computing
    → symmetric multiprocessor computing
    → massively parallel computing

- **Grid computing** – distributed system with non-interactive workloads
  * each node performs different task
  * trust and security between domains (resource sharing agreements)
  * no centralized control
  * std, open protocols and interfaces
  * std for auth, resource delivery, access
  * virtual organizations
  * data grids: BIRN, SCEC, Science grid



- **Cloud computing** – centralized / distributed computing system
  * efficiency
  * dependability
  * adaptation
  * flexibility

Cloud computing Models

- Deployment models
  1. Private
  2. Public
  3. Hybrid

- Tech that enables CC
  1. Broadband networks
  2. Data center tech
  3. Virtualization tech
  4. Web tech
  5. Multitenant tech

- Cloud service models
  1. IaaS: AWS, Azure, Google Cloud, IBM Cloud, Oracle Cloud
  2. PaaS: Google Apps Engine, Azure, Force.com
  3. SaaS: CRM, Office Suite, Google Apps, email

Technology challenges

1. Scalability
2. Elasticity
3. Performance unpredictability
4. Reliability
5. Availability
   - fail detect - heartbeats
     - probes
   - app recovery - redirection
6. Security
7. Compliance
8. Multi-tenancy

Business Drivers

- Operational
- Cost

Cloud Service Models

1. Public Cloud

- Shared resource allocation
- Usage agreements
- Management
- Advantages
  * cost
  * scalability
  * analytics
- Disadvantages
  * security
  * compliance
  * vendor lock-in

2. Private Cloud

- In-house infra
- Internal vs hosted
- Disadvantages
  * cost
  * under-utilization
  * scaling

3. Hybrid Cloud

- Use both for different tasks

Distributed System Models

1. Architectural Models

(i) System Architecture

- P2P
- Client-server

(ii) Software Architecture

- 3-tier architecture (database, business logic, presentation)

2. Interaction Models

- Synchronous DS (shared clock
- Asynchronous DS

3. Fault Models

- Omission
- Arbitrary
- Timing
- Types
  * transient
  * intermittent
  * permanent

CLOUD ARCHITECTURE

1. Front-end
   - Client-side interfaces to cloud
   - Web, mobile, tablet

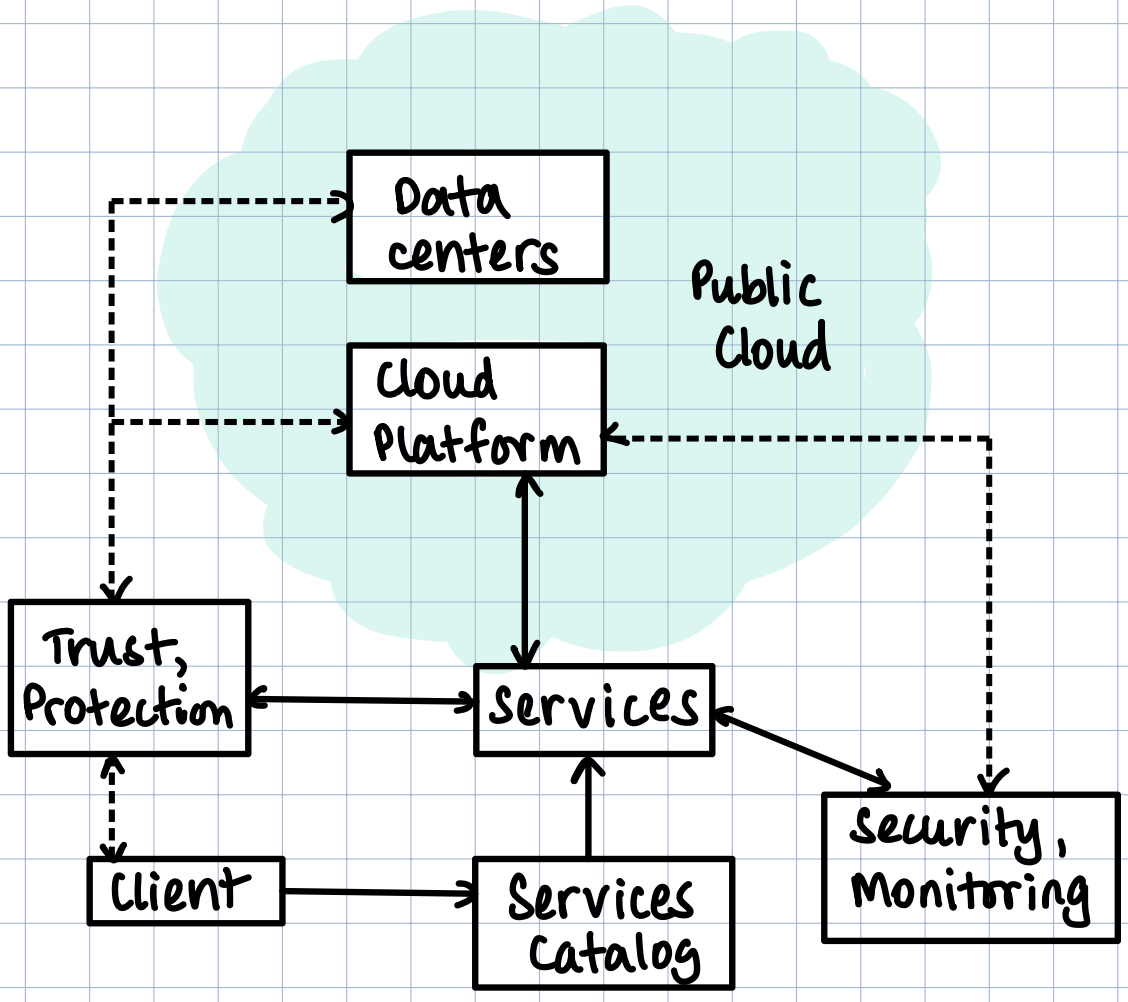
2. Back-end
   - Used by service provider
   - Manages resources
   - Components
     * Application (sofware, platform)
     * Service (enables IaaS, PaaS, SaaS)
     * Runtime cloud
     * Storage
     * Infrastructure
     * Management
     * Security

3. Network
   - Internet
   - Bridge b/w front & back

CLOUD PLATFORM DESIGN GOALS

1. Scalability
2. Efficiency
3. Reliability & Availability
4. Simplifying UX

Generic Cloud Architecture



Market-Oriented Architecture

- Regulates allocation based on demand (QoS and economic)
- Request examiner: ensures no over-provisioning
- Pricing mechanism
- VM monitor mechanism
- Dispatcher mechanism: starts execution of reqs on VMs
- Service request monitor mechanism: keeps track of execution progress

CLOUD SERVICE MODELS

Execution Model

- How programs in a language are executed

Programming Model

- Execution model linked into an API
- Programming model of single system: execute C code instruction-by-instruction, use disk, memory, CPUs
- Cloud environments: execution model of language same, additional execution model of programming model
- Disk, memory, CPU not guaranteed in distributed systems; programming model must account for it

1. IaaS

- Provided by service: compute, storage
- Consumer control: OS, storage, deployed apps, limited control on networking components
- Compute: Sole ownership of VM / container
- Storage: block, file, object

- Networking: software defined networking (APIs)
- Eg: AWS EC2, Azure VMs, Google compute engine, IBM Cloud Private

2. PaaS

- Provided by service: underlying hardware, OS, middleware
- Consumer control: application, databases
- Advantages
  * FTTM
  * Multiple platforms
  * Scalable
- API development and management
- IoT
- Eg: AWS EB, Azure DevOps, Google App Engine
- Watch AWS Elastic Beanstalk video

3. SaaS

- Access software from client
- Google Apps - Docs, Sheets etc
  MS Apps - office 365
  File storage - OneDrive, Google Drive, DropBox

Email - Outlook, Gmail
CRM - SalesForce

- Characteristics
  * Multi-tenant architecture
  * Easy customization

Service-Oriented Architecture

- Make software components reusable via service interfaces

- Loose coupling; exposed with std network protocols

- Two SOA styles (for web services)
  * REST
  * SOAP

(1) REST - REpresentational State Transfer

- Stateless, reliable APIs (stateless — servers can change)

- Mandatory constraints
  * Separation of concerns / client-server constraint
  * Stateless constraint (state Transfer of REST)
  * Cache constraint (data within response marked as cacheable or not)
  * Uniform interface constraint
    ↳ Resource & resource identification (URI)
    ↳ Manipulation of resource through representations
    ↳ self-descriptive messages
    ↳ Hypermedia

* Layered system constraint

- Client requests for resource through URI, server responds with representation of resource (hypermedia)
- Safe – no modification – GET
- Idempotent – no effect if called repeatedly with same input – GET, PUT, DELETE
- Multiple representations ; metadata

(2) SOAP - Simple Object Access Protocol

- Before REST
- XML transmission over SMTP, HTTP, FTP
- Message [envelope [header [auth], body [payload]]]
- Web services using SOAP
  * UDDI : registry for businesses to list themselves on the internet
  * WSDL: web services description language

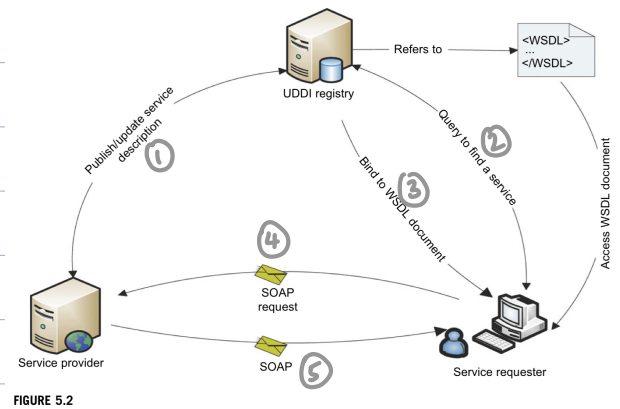


FIGURE 5.2
A simple web service interaction among provider, user, and the UDDI registry.

COMMUNICATION between PROCESSES

1. Synchronous

- Request-response
- Blocking
- Timeout

2. Asynchronous

- Pub-sub
- Message queues
- Event-based
- Batch processing
- store and forward
- Advantages
  * Reduced coupling
  * Multiple subs
  * Failure isolation
  * Load leveling
- Disadvantages
  * Latency
  * Coupling with infra
  * Complexity
  * Throughput

(a) One-to-one

→ Async req-res

→ One way notifs

(b) One-to-many

→ Pub-sub

→ Pub-async responses

Message Queues

- Messages stored on queue until processed & deleted (buffer)
- Producer adds message, consumer retrieves message and processes
- One-to-one (each message processed only once)
- Can be combined with pub-sub

Publisher - Subscriber

- Message published to a topic received by all of the subscribers to the topic
- core concepts
  * Topic
  * Message
  * Publisher
  * Subscriber
- Advantages
  * Loose coupling
  * Scalability
  * Eliminate polling
  * Dynamic targeting
  * Decouple and scale independently
  * Simplify communication

Redis

- In-memory k-v datastore used as queue, cache, database, message broker

APPLICATION ARCHITECTURE

1. Monolithic

- Single unit (indivisible)
- Simple, not scalable

2. Microservices

- Small, collaborative services
- Lightweight communication
- Benefits
  - * Flexibility
  - * Reliability
  - * Development speed
  - * Complex apps
  - * Scalability
  - * Continuous deployment
- Principles
  - * Single Responsibility
  - * Modeled around business domain
  - * Isolate failure
  - * Infra automation (scripting environment)
  - * Deploy independently
- Evolution of SOA

Migrating Monolithic to Microservice

- Migrate to cloud - 6 R's
  - * Re-host (lift-and-shift; migrate to cloud w/o changes)
  - * Re-platform (lift-tinker-and-shift; optimizations made)
  - * Re-architect (highest ROI; remodeling; SOA or μservice)
  - * Re-purchase (move perpetual licenses to saas)
  - * Retire (remove features not in use)
  - * Retain (keep critical features)

post migration

- Challenges
  - * Service decomposition (find seams, use containers)
  - * Persistence (monolith db decomposition)
    - ◆ Reference table segregation options
      - ↳ data as an API
      - ↳ projection / replication of data
    - ◆ Shared mutable data
      - ↳ as a separate μservice
    - ◆ Shared table
      - ↳ split
  - * Transaction boundaries (Two phase commit, compensating transactions)
  - * Performance
  - * Testing
  - * Inter-service communication

Suitability	Rehosting	Replatforming	Rearchitecting
Application suitability	• Web compatible UI • MVC architecture • Flask, Symphony, .NET can be rehosted without complications • Applications not built with web frameworks but have a web-compatible UI (HTML, PHP) not ideal	• Web-compatible UI and MVC architecture • Must allow easy connection to the database server • Traditional apps using Xampp-like servers are suitable	• Monolithic architecture, but are web-compatible are suited • Large codebases • Resource intensive • Desktop applications built using frameworks are not suited
Business suitability	• Higher network speed • Low technical debt and cost of maintenance. • Small apps that require a temporary boost in performance	• Database hosting most common • Improve DB scalability • Boosts fault tolerance • Data is more resilient.	• Looking for large improvements in availability, testability, continuous delivery, reusability and lower infrastructure costs