Introduction to Cloud Applications

Introduction

1. High level overview in best practices for developing scalable applications for the cloud
   - Evaluation pros and cons of migrating applications from enterprise into the cloud

Developing Large Applications for the Cloud

1. App developer's toolkit on the Cloud
   - Frameworks
     - json, j2ee, .NET, REST
   - Webserver
     - Apache MS IIS, IBM HIS
   - Appserver
     - Websphere, Weblogic
   - Database Server
     - MySQL, SQL, DB2, Oracle
   - NoSQL Databases
     - MongoDB, Hadoop
   - Development Environment
     - Eclipse, Rational, Visual Studio
   - Configuration/Deployment
     - Chef, Puppet
   - Lifecycle Management
     - CVS, Subversion, Git
   - Testing
     - LoadRunner, Rational

Best Practices for Scalable Cloud Application Development

1. Stateless design
   - Don't store state on the server so clients can communicate with any server
2. Loose coupling of app components
   - Don't share state among components
3. Asynchronous communication
   - Failures can occur, so calls should be asynchronous so the client can query the health of any calls that were made
   - Promote parallelism
4. Choice of Database
   - If application doesn't require semantics of a SQL database, a NoSQL database is likely preferable
5. Design patterns
   - Many applications fall into similar well-defined patterns, so cloud providers provide these patterns instructing how to structure a service
     - For example, a shopping cart
   - Promotes reuse of components

Best Practices for Ensuring Reliability and Availability

1. Ensuring no single point of failure
   - Redundancy in MapReduce using Zookeeper
2. Automate failure actions
   - Don't want to have to manually address every failure
3. Graceful degradation
   - Elastically increase resources, might not be able to provide service at the same level

4. Liberal use of logging
    - Understand what happened when things go wrong
    - Ensure that reliability logic is not on the critical path so it doesn't decrease performance
5. Replication
    - Key for ensuring reliability and availability
    - Partition dataset so workers can process in parallel
    - Some datasets aren't partition-able, such as Gmail inbox
    - Search is a good candidate for partitioning

Best Practices for Ensuring Security

1. Authentication
    - Are clients who they say they are?
2. Authorization
    - Identity management and access management
    - Are clients allowed to do what they're trying to do?
3. Data Security
    - Stored and in-flight data
4. Encryption and Key management
    - Important for ensuring data security
5. Auditing
    - Want to be able to understand what happened when something goes wrong

Best Practices for Ensuring Performance and Endurance

1. Performance
    - Identify metrics of importance
        - Response time, throughput, false positives, false negatives
    - Workload modeling
        - Need a representative dataset for stress testing
    - Stress testing
        - Identify bottlenecks
2. Endurance
    - Design for testing and verification
        - Built in mechanisms for independently measuring the health of components
        - Automated mechanisms for gathering and reporting health and performance statistics
    - Design for future evolution
        - Incremental addition of new features
        - Incremental testing of upgrades
        - Automated maintenance

New Normal

1. Confluence of several factors
    - Distributed computing
    - Grid computing to democrative HPC use by scientists
    - WWW
    - Gigabit networks
    - Appearance of clusters as a parallel workhorse
    - Virtualization
    - Enterprise transformation
    - Supply chain
    - B2B commerce
    - Business opportunity for computing as a utility service

2. Concrete evidence of success
    - Driving down costs and accuracy
        - FDA: Manual report to machine readable reports
            * 99.7% accuracy
            * Cost down from $29 per page to $0.25 per page
    - Data security
        - Organizations can benefit from the "sledge hammer" of Cloud service providers rather than internal IT teams
    - Spurring innovation
        - Cost of experimentation is small

Designing for Scale

1. Gap between expectation and reality
    - Expectation: Cloud resources are infinite
    - Reality: Cloud resources have finite capacity
        - Number of CPUs, network bandwidth, etc.
2. Implications
    - Design for "scale out"
    - Capacity planning is crucial
    - Carefully understand "scale unit" for your app
        - Multiple smaller deployments
        - E.g., one instance for every 100 core
    - Automate provisioning resources commensurate with the scale unit for your app
3. A real work example
    - Azure's solution for financial risk modeling
    - Pathological example
        - Cost of insuring the entire world population
            * Model: Stochastic analysis of the insurance cost
            * Simulation time: 19 years on a single core
        - Azure's solution
            * Without changing the programming model runs on 100,000 geo-distributed cores

Challenges of Migrating to the Cloud

1. Service parameters
    - Latency, uptime/availability
    - Security
    - Legal issues
2. Enterprise services are complicated
    - N-tier (front-end, business logic, back-end)
    - Replication and load balancing
    - Intra-org firewalls
3. Solution
    - Hybrid implementation
        - Private plus public
    - Construct a model of cost/benefit of migrating each component

Conclusion

1. Cloud is not a panacea for instant scalability and performance of enterprise applications
    - Migrating and architecting an app for the cloud requires considerable thought
    - Always keep scale up as the fundamental design principle
    - Proliferation of cloud resources is a testament to cloud computing becoming the new normal