Cloud Computing with Microsoft Azure

Michael Stiefel
www.reliablesoftware.com
development@reliablesoftware.com

© Copyright 2009 Reliable Software, Inc.
Azure's Three Flavors

Azure Operating System (*Platform as a Service*)
- Worker/Web Role, Blobs, Queues, Tables

Azure .NET Services (*Software as a Service*)
- Access Control Service
- SQL Azure (SQL Server in the sky)
- Workflow Services

Azure Hosted Services (*Application as a Service*)
- Hosted Exchange
- Host SharePoint

© Copyright 2009 Reliable Software, Inc.
Cloud Operating System

Abstracts the underlying infrastructure
Manages resources
Azure Platform

Service management

Compute

Storage

Developer experience

You define rules and provide code

Platform deploys, monitors, and manages your service according to your rules
Azure Services Platform

Your Service

Web/ Worker Role
Blobs, Tables, Queues

SQL
Azure

Access Control

Live Services

Dynamic CRM Services
Demo: Scalable Architecture

- Cloud Storage (blob, table, queue)
Cloud Computing is Utility Computing

- Illusion of Infinite Computing Resources on Demand
- No up front commitment
- Pay for resources as needed
Three Classes of Vendors

Scalability, Failover, Recovery

Amazon
Google / Force.com
Microsoft
Utility Computing Scenarios

Outsource Your Infrastructure
Occasional Need for Massive Computation
No Need to Build to Peak Capacity
Cloud-Bursting
Software as a Service
Data Close To Your Customer
Internet Scale
Economic Conditions

Pricing

Service Level Agreement (SLA)
Azure Platform Pricing

Compute $0.12 per hour
Storage $0.15 per GB month
Storage Transactions $0.01 per 10K
Bandwidth
  $0.1 in per GB
  $0.15 out per GB
Within the datacenter is free
SQL Azure

Up to 1 GB database $9.99 /month
Up to 10 GB database $99.99 / month

Bandwidth

0.1 in per GB
0.15 out per GB
SMB Data Costs

10 GB SQL Database
2 GB a month data in, 4 GB a month data out

$100.77 a month
A SAN can cost from $30-40,000
25 year equivalent

Does not consider the cost of infrastructure employees or the software licenses.
SMB Compute Costs

$1051 per year for one compute process with no idle time

$31.53 if you did a storage save every second

$3600 per year 2 TB of disk storage

About $5000 / year

Employee and licensing costs not considered
Announced Azure SLA

Computation: 99.95% up time
SQL Azure: 99.9% up time
## Utility SLA

<table>
<thead>
<tr>
<th>Service</th>
<th>2007</th>
<th>2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calls Answered Within 30 Seconds</td>
<td>Goal</td>
<td>Actual</td>
</tr>
<tr>
<td></td>
<td>80%</td>
<td>84.64%</td>
</tr>
<tr>
<td>Average # Service Interruptions Per Customer</td>
<td>1.373</td>
<td>1.027</td>
</tr>
<tr>
<td>Average # Min Without Power Per Customer</td>
<td>168.69</td>
<td>82.61</td>
</tr>
<tr>
<td>Service Appointments Met</td>
<td>87.78%</td>
<td>98.52%</td>
</tr>
<tr>
<td>Actual Meters Read &quot;on cycle&quot; vs estimate</td>
<td>93.15%</td>
<td>98.75%</td>
</tr>
<tr>
<td>Complaint Cases Per 1000 Customers</td>
<td>1.496</td>
<td>.974</td>
</tr>
</tbody>
</table>

**Utility Availability:** 99.98%
Compelling Case

SMB Applications
Massive Computation Needs
No Need to Build to Peak Capacity
Cloud Bursting
Software as a Service
Utility Computing Scenarios

Outsource Your Infrastructure
Occasional Need for Massive Computation
No Need to Build to Peak Capacity
Cloud-Bursting
Software as a Service
Data Close To Your Customer
Internet Scale
Latency Exists

Speed of light in fiber optic cable: 124,000 miles per second

A ping Japan from Boston takes 100 ms.
Real number is about 250 ms.
Fetch 10 images for a web site: 1 second
Ignores Latency of the operation
Bandwidth is Limited

Shannon's Law: $C = B \log_2 (1 + S / N)$
Capacity = bit / second
Bandwidth (hertz)
S/N * 5 to double capacity given bandwidth
Latency is Not Bandwidth

Size of the shovel vs. how fast you can shovel

Infinite shovel capacity (bandwidth) is limited by how fast one can shovel (latency).
Great Bandwidth, Poor Latency

Buy a two terabyte disk drive
Put it in a car and drive to New York
Expensive to Move Data

Computational Power Gets Cheaper Faster than Network Bandwidth

Cheaper to compute where data is instead of moving it

*Distributed Computing Economics* Jim Gray

Want data to be close to where your customer is...
Connectivity is Not Always Available

Cell phone
Data Center Outages
Equipment Upgrades
Data redundancy to improve reliability
Waiting for Data Slows Computation

Partition Your Data to Improve Performance
Partition Your Data to Achieve Internet Scale
Data Naturally Lives In Multiple Places
Distributed Transactions Impede Throughput
Human Interaction
Relational Databases Scale Up Not Out

Relational Databases scale well on a single node or cluster

- Complexity of relations
- Query plans with hundreds of options the query analyzer evaluates at runtime
- Normalization
- ACID Transactions

Two Phase Commit guarantees consistency if you have infinite time

Quick scale up difficult with hardware upgrade
Economics Dictate Scale Out Not Up

Cheap, commodity hardware argues for spreading load across multiple servers

How do you distribute data among several databases?
How do you achieve consistency?
How do you achieve throughput with distributed transactions?
CAP Theorem

Can Have Any Two

Eric Brewer, UC Berkeley, Founder Inktomi
Consistency and Availability

- Single site Database
- Database Cluster
- LDAP
- Two phase commit
- Validate Cache
Consistency and Partitioning

- Distributed Database
- Distributed Locking
- Pessimistic Locking
- Minority Partitions
- Invalid
Availiability and Partitioning

- Forfeit Consistency
- Google BigTable
- Amazon Simple DB
- Optimistic
- Can Denormalize
- No ACID transactions
- Compensation
Storage in Azure

World of Consistency

SQL Azure

World of Internet Scale (Numbers or Geography)

Blobs, Tables, Queues
SQL Data Services

Revised to be SQL Server in the sky
  Tables, Stored Procedures, Triggers, Constraints Views, Indices
  Uses TDS (Tabular Data Stream) Protocol

Change connection string to get to another SQL Server

No Current Availability
  • Get Started with SQL Express
Windows Azure Storage Services

Tables of key/value pairs for highly scalable structured storage

CRUD operations

No FK relations, Joins, Constraints, Schemas

Partition / Tables / Entities / Properties

Entity has Unique Row Key
Azure Storage Services

Fit well with tens or hundreds of commodity servers
Better mapping with objects than ORM
No integrity constraints
No joined queries
No standards among vendors (lock in)

Will Microsoft have query limits?
  Amazon no query longer than 5 seconds
  Google no more than 1000 items returned
## Car Table

<table>
<thead>
<tr>
<th>Key</th>
<th>Attribute 1</th>
<th>Attribute 2</th>
<th>Attribute 3</th>
<th>Attribute 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Make: BMW</td>
<td>Color: Grey</td>
<td>Year 2003</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Make: Nissan</td>
<td>Color: Red Yellow</td>
<td>Year: 2005</td>
<td>Transmission: Easytronic</td>
</tr>
<tr>
<td>3</td>
<td>Plane: Boeing</td>
<td>Color: Blue</td>
<td></td>
<td>Engine: Rolls Royce</td>
</tr>
</tbody>
</table>


Do You Need To Partition Your Data to Scale?

No Partitioning
Natural Partitioning
Partitioning for Availability
If you have to partition to scale, how do you decide between availability and consistency?
What is the Cost of an Apology?

Amazon
Airline reservations
Stock Trades
Deposit of a Bank Check
Deleting a photo from Flickr or Facebook
Sometimes the cost is too high

Authentication
  SAML tokens expire

Launching a nuclear weapon
Businesses Apologize Anyway

Vendor drops the last crystal vase
Check bounces
Double-entry bookkeeping requires compensation at least 13th century
Eventually make consistent
State of the Software != State of the World
Software approximates the state of the world
It makes the best guess possible
Sometimes that is wrong
Other computers might have other opinions
Overturn software myths of the past 25 years.
How consistent?

Business Decision
How much does it cost to get it absolutely right?
Computers can remember their guesses
Can replicate to share guesses
It may be cheaper to forget, and reconcile later
Design For Eventual Consistency

Identify objects by unique key (partition key / row key)

Objects can move when repartitioning

Cannot assume two objects remain on the same machine

Data might go offline

Transactions can only apply on per object basis

Different computations might come to different conclusions

Define message based workflows for ultimate reconciliation and replication of results
Demos ?
Conclusions

Understanding Azure is about understanding
Economics of cost and availability
Need for Scalability
Architectural Implications
Design for Eventual Consistency
Remember the 2 / 10 rule