SQL Azure: Database-as-a-Service
What, how and why Cloud is different

Nigel Ellis <nigele@microsoft.com>
July 2010
Talk Outline

• Database-as-a-Service
• SQL Azure
  – Overview
  – Deployment and Monitoring
  – High availability
  – Scalability
• Lessons and insight
SQL Azure Database as a Service

- On-demand provisioning of SQL databases
- Familiar relational programming model
  - Leverage existing skills and tools
- SLA for availability and performance
- Pay-as-you-go pricing model
- Full control over logical database administration
  - No physical database administration headaches
- Large geo-presence
  - 3 regions (US, Europe, Asia), each with 2 sub-regions
Challenges And Our Approach

- **Challenges**
  - Scale – storage, processing, and delivery
  - Consistency – transactions, replication, failures, HA
  - Manageability – deployment and self-management

- **Our approach**
  - SQL Server technology as node storage
  - Distributed fabric for self-healing and scale
  - Automated deployment and provisioning (low OpEx)
  - Commodity hardware for reduced CapEx
  - Software to achieve required reliability
- Each **account** has zero or more **servers**
  - Azure wide, provisioned in a common portal
  - Billing instrument

- Each **server** has one or more **databases**
  - Zone for authentication: userId+password
  - Zone for administration and billing
    - Metadata about the databases and usage
  - Network access control based on client IP
  - Has unique DNS name and unit of geo-location

- Each **database** has standard SQL objects
  - Unit of consistency and high availability (autonomous replication)
  - Contains Users, Tables, Views, Indices, etc...
  - Most granular unit of usage reports
  - Three SKUs available (1GB, 10GB and 50GB)
ARCHITECTURE
Applications use standard SQL client libraries: ODBC, ADO.Net, PHP, JDBC, ...

Load balancer forwards ‘sticky’ sessions to TDS protocol tier

Gateway: TDS protocol gateway, enforces AUTHN/AUTHZ policy; proxy to SQL tier

Scalability and Availability: Fabric, Failover, Replication, and Load balancing
HIGH A VAILABILITY
**Concepts**

**Storage Unit**
- Supports CRUD operations
- e.g. DB row

**Consistency Unit** (aka Rowgroup)
- Set of storage units
- Specified by “application”
- Range partitioned or entire DB
- SQL Azure uses entire DB only
  - Infra supports both

**Failover Unit** (aka Partition)
- Unit of management
- Group of consistency units
- Determined by the system
- Can be split or merged at consistency unit boundaries
Data Consistency

• Each Failover Unit is replicated for HA
  – Desired replica count is configurable and actual count is dynamic at runtime

• Clients must see the same linearized order of read and write operations

• Replica set is dynamically reconfigured to account for member arrivals and departures
  – Read-Write quorums are supported and are dynamically adjusted
Replication

- All reads are completed at the primary
- Writes replicated to write quorum of replicas
- Commit on secondaries first then primary
- Each transaction has a commit sequence number (epoch, num)
Reconfiguration

- Types of reconfiguration
  - Primary failover
  - Removing a failed secondary
  - Adding recovered replica
  - Building a new secondary

- Assumes
  - Failure detector
  - Leader election
  - Both services provided by Fabric layer

Safe in the presence of cascading failures
Partition Management

• Partition Manager (PM) is a highly available service running in the Master cluster
  – Ensures all partitions are operational
  – Places replicas across failure domains (rack/switch/server)
  – Ensures all partitions have target replica count
  – Balances the load across all the nodes

• Each node manages multiple partitions

• Global state maintained by the PM can be recreated from the local node state in event of disaster (GPM rebuild)
System in Operation

Primary Master Node

- SQL Server
- Global Partition Map
- Partition Manager
  - Partition Management
  - Partition Placement Advisor
- Leader Elector
- Load Balancer
- Fabric

Data Node 100
- P
- S
- S
- S
- S

Data Node 101
- S
- S
- S
- P

Data Node 102
- P
- S
- S
- S

Data Node 103
- P
- S
- S
- S

Data Node 104
- S
- P
- S
- S

Data Node 105
- P
- S
- S
- S

Fabric

Secondary → Primary → Secondary
SQL node Architecture

- Single physical DB for entire node
- DB files and log shared across every logical database/partition
  - Allows better logging throughput with sequential IO/group commits
  - No auto-growth on demand stalls
  - Uniform manageability and backup
- Each partition is a “silico” with its own independent schema
- Local SQL backup guards against software bugs
Recap

• Two kinds of nodes:
  – Data nodes store application data
  – Master nodes store cluster metadata

• Node failures are reliably detected
  – On every node, SQL and Fabric processes monitor each other
  – Fabric processes monitor each other across nodes

• Local failures cause nodes to fail-fast

• Failures cause reconfiguration and placement changes
DEPLOYMENT
Each rack hosts 2 pods of 20 machines each
Each pod has a TOR mini-switch
  • 10GB uplink to L2 switch
Each SQL Azure machine runs on commodity box
Example:
  • 8 cores
  • 32 GB RAM
  • 1TB+ SATA drives
  • Programmable power
  • 1Gb NIC
Machine spec changes as hardware (pricing) evolves
Hardware Challenges

• SATA drives
  – On-disk cache and lack of true "write through" results in Write Ahead Logging violations
    • DB requires in-order writes to be honored
    • Can force flush cache, but causes performance degradation
  – Disk failures happen daily (at scale), fail-fast on those
    • Bit-flips (enabled page checksums)
    • Drives just disappear
    • IOs are misdirected

• Faulty NIC
  – Encountered message corruption
    • Enabled message signing and checksums
Software Deployment

• OS is automatically imaged via deployment
• All the services are setup using file copy
  – Guarantees on which version is running
  – Provides fast switch to new version
  – Minimal global state allows running side by side
  – Yes, that includes the SQL Server DB engine
• Rollout is monitored to ensure high availability
  – Knowledge of replica state health ensure SLA is met
  – Two phase rollouts for data or protocol changes
• Leverages internal Autopilot technologies with SQL Azure extensions
Software Challenges

• Lack of real-time OS features
  – CPU priority
    • High priority for Fabric lease traffic
  – Page Faults/GC
    • Locked pages for SQL and Fabric (in managed code)

• Fail fast or not?
  – Yes, for corruption/AV
  – No, for other issues unless centrally controlled

• What is really considered failed?
  – Some failures are non-deterministic or hangs
  – Multiple protocols / channels means partial failures too
Monitoring

- Health model w/repair actions
  - Reboot → Re-deploy → Re-image (OS) → RMA cycle
- Additional monitoring for SQL tier
  - Connect / network probes
  - Memory leaks / hung worker processes
  - Database corruption detection
  - Trace and performance stats capture
    - Sourced from regular SQL trace and support mechanisms
    - Stored locally and pushed to a global cluster wide store
    - Global cluster used for service insight and problem tracking
LESSONS LEARNED
How is Cloud Different?

Minor differences:

• Cheap hardware
  – No SANs, no SCSI, no Infiniband
  – Iffy routers, network cards
  – Relatively homogeneous
  – *Hardware not selected for the purpose*

• Lots of it
  – Not one machine, not 10 machines – think 1000+

• Public internet
  – High latencies, sometimes
  – All over the world
  – Scary people (untrusted) lurking in the shadows
How is Cloud Different?

Real differences:

• You are responsible for the whole thing
  – No such thing as “can you send us a repro”
  – No such thing as “it’s a hardware problem” (it’s us)
  – No such thing as “it’s a network issue” (it’s us)
  – No such thing as “it’s a configuration issue” (it’s us)
  – No such thing as “It’s not us, it’s DNS” (it’s us)
  – No such thing as “It’s not us, it’s AD” (it’s us)

• User expectations: it’s a utility!
  – Utility of databases, not instances or servers
  – Highly available (means “it’s there” not “replication has been enabled”)
  – Elastic (you need more, you can have it right away)
  – Load-balanced (automatically)
  – And yet: symmetric (“give me cursors or give me death”)
Design for Failure

• Common mistake #1: Failures can be eliminated
  – Everybody fails! Hardware, software, universe

• Common mistake #2: All failures can be detected
  – No watchdog is fast enough or good enough

• Common mistake #3: Failures can be enumerated
  – Cannot deal with issues one at a time
  – Must take a holistic, statistical approach
  – Learn only as much as you need to take action

• Common mistake #4: Failures can be dealt with independently
  – Local observation generates insufficient insight, leads to global disasters
Design for Failure

1. Observe and detect
2. Collect context
3. Make decisions
4. Aggregate complaints
5. Commit decisions
6. Send complaints
7. Implement decisions

Local vs. Centralized
Design for Mediocre

• **Network** is not fast or slow, it varies
  – Design for huge latency variance
  – Machine independence is key

• **Machines** are not up or down, they are kind of slow
  – Measure, it’s never black-and-white

• **People** are not good or fired, they all make mistakes
  – Tools and processes to minimize risk

• **Environment** is often iffy
  – Integrated security? Not so fast...

• It’s less important to succeed, than to **know the difference**
Design for (appropriate) Simplicity

• There’s no such thing as a “repro”
  – Everything must be debuggable from logs (and dumps)
  – This is much harder than it sounds – takes time to log the right stuff
• System state must be externally examinable
  – Not locked in internal data structures
• Fail-fast
  – Is great! Very hard to reason about partial failures. We kill it fast.
  – Is awful! Cascading failures can kill entire system if you are not careful
  – Principle: If you are sure it’s local, kill it. If not, not so fast
• ‘No workflows’ is best
  – Machine independence is a virtue
  – Things that can safely be local, should be
• Single-level workflows is next (reduce number of moving parts)
  – Resumable (not tied to a specific machine)
  – Design with failure as norm using distributed (persisted) state machines
Design for many

• Many machines is great!
  – Reduce focus on machine **reliability**
    • By the time RDBMS runs recovery, the world has moved on
  – Distribution enables **load-balancing**
    • Focus on elasticity and flexibility
  – **HA with 100 machines is better than 2**
    • Load distribution, parallelism of copy

• Many machines is hard!
  – Elasticity needs to be **built** in
  – All operations must be **multi-machine**
  – Correlated failures are a fact of life
Design for multi-tenancy

- Customers like using many machines
  - Enables load-balancing and elasticity
  - But they don’t like paying for many machines

- Solution: multi-tenancy!
  - Everyone gets many slices

- Hard!
  - Isolation for security and performance
  - Many small databases? Costs....
  - Many relationships (replication)
  - Tradeoffs: isolation vs. elasticity?
Balance between local and global is key!

• “Normal case” decisions must be local
  – Any global state (e.g. routes) must be cached
  – The fewer parties are involved, the better
  – Otherwise: bottlenecks, single points of failure

• “Special case” decisions must be global
  – How to react to an error?
  – When to failover?
  – When and where to balance load?
  – When and how to upgrade software?
  – Otherwise: instability, chaos, low availability

• Data must be where it is needed
  – Global data needed for local operations must be cached locally
  – Local data needed for management must be aggregated globally
Real Symmetry is End-to-End

• Symmetry is **not** just about **surface** area
  – Too much focus on features

• It’s **not** symmetric if:
  – If the **syntax** is the same, but it works in subtly different ways
  – If my connection **drops** too often
  – If the **latency** causes me to put everything in SPs
  – If operations unpredictably take **10x** as long sometimes

• Customers want clarity, **predictability**, and minimal **learning curve**
Summary

• Cloud is different
  – Not a different place to host code

• Opportunities are great
  – Customers want a utility approach to storage
  – New businesses and abilities in scale, availability, etc

• But the price must be paid
  – Which is a good thing, otherwise everyone would be doing it!
Future Work and Challenges

• Performance SLAs
  – Delivering on “guaranteed capacity” while consolidating diverse workloads is hard

• Privacy, Governance and Compliance
  – Perceptions and realities
  – Private Cloud appliances

• Programming Models
  – Support for loosely coupled scaleout patterns such as sharding
  – Transparent multi-node scaleout

• Data Redundancy
  – Point in time restore (backup knobs)
  – Geo-availability for multiple points of presence

• Health Model for Applications
  – Data tier is only part of the problem – support for hosting N-tier apps and providing insight into health and performance
QUESTIONS?
SQL Azure Links

• SQL Azure
  http://www.microsoft.com/windowsazure/sqlazure/

• SQL Azure “under the hood”
  http://www.microsoftpdc.com/sessions/tags/sqlazure

• SQL Azure Fabric