SQL Azure as a Self-Managing Database Service: Lessons Learned and Challenges Ahead
1 Introduction

Key features:
- Shared nothing architecture
- Log based replication
- Support for full ACID properties
- Consistency and high availability
- Near 100% SQL Server delivered programming model at cloud scale
Type of databases used, application contexts, primary customer expectations.

- OLTP consolidation is a predominant category (non-OLTP as well)
  - Database resource requirements are smaller than a single node in the cluster
  - Relatively small nodes are used
  - Customers want to pay as they go
  - Customers want and get a good deal via commodity hardware, automatic management of core pieces of the system, and high availability.
- System must handle most faults without operator intervention
Self management and fault tolerance permeate each level

Protocol Gateway

- Enables client to connect to SQL Azure service
- Coordinates with distributed fabric to locate the primary, and renegotiates that choice in case a failure or system-initiated reconfiguration causes election of a new primary

Database Engine

- Manages multiple locally attached disks, each private to that instance.
- If a disk fails, watchdog processes which monitor hardware health will trigger appropriate failure recovery actions.
Self management and fault tolerance permeate each level

Distributed Fabric Layer
- Maintains up/down status of servers
- Detects server failure
- Orchestrates recoveries

Global Partition Manager
- Maintains a directory of information about partitions
- If a primary dies the GPM will elect one of the remaking secondary nodes to become a primary
Self management and fault tolerance permeate each level

Infrastructure and Deployment Services
- Responsible for hardware and software lifecycle management and monitoring
- Hardware lifecycle includes preparing metal machines for a role on the cluster, monitoring machine health, taking a machine through repair cycles: reboot, reimage, return to manufacturer (RMA)
- Software lifecycle runs workflows to clean a cluster, specify machine roles, deploy application software, deploy OS patches
Self-healing Self-managing.

- Cloud database systems need self-healing capabilities in order to scale to large numbers of clusters and large number of machines in each cluster.
- Current SQL self-managing capabilities are in the following areas:
  - Detects hardware failures and converts these high severity exceptions into node failures.
  - System managed backups.
  - Periodic snapshots of SQL Server state on each node that are used for trending and debugging.
  - Throttling to prevent user requests from overwhelming the machine.
2 Lessons in Self-Management and Fault Tolerance
Define the metrics to instrument and monitor the cluster effectively.

- Everything in hardware and software fails sooner or later, and sometimes in bizarre and mysterious ways!

- The right mix of health metrics should be detailed, provide aggregate views across logical and/or physical groups of machines, and be able to drill effectively into failure cases and hotspots.
Take a calibrated approach to dealing with serious failures.

- Example – serious software issue like an Access Violation are typically highly localized to a single machine and in response to a very specific mix of workloads and may cause serious consequences like index corruption.

- Rule of thumb – fail fast when the failure conditions are localized.

- However failing fast is not always the best option.

- Incorrect system configuration could cause cascading failures because configurations are common across many machines.

- If scope of failure exceeds some standard deviation (automation can help figure this out) the policy is to get humans involved.

- Big Red Switch is a mechanism for turning off automatic repair or compensation operations.

- Statement Firewall disallow combinations of input data and sql statements already know to cause failure.
Software-based defense in depth is appropriate when building on commodity hardware.

SQL Azure was built on commodity hardware to provide an attractive price point for customers therefore must invest more in software for higher service reliability.

- Things needed to support software as our defense in depth strategy
  - Using checksums to verify data on disks
  - Periodically sampling data on disk for disk rot
  - Ensuring log handshakes are honored
  - Signing all network communications to guard against bugs
  - NIC firmware
  - String information (such as identity metadata) redundantly, allowing to reconstruct / rebuild in case of catastrophic failure
  - Enforce homogeneity across the cluster
Build a failure classifier and playbook.

- Let each failure teach you something for the future
- Some of the most common failure types

1. Planned failures – application software upgrades, OS patches, firmware upgrades, and planned load balancing events, mediated by the IDS layer
2. Unplanned failures – Hardware and network errors, software bugs, human error

- Resolved with two phase upgrade, deconstructing a monolithic service or major component into several smaller services, software based defense in depth policy, mitigate first fix next policy
Design everything to scale out.

- Favor decentralized service architecture and avoid choke points
- GPM only centralized piece of the SQL, it is a scaled out single point of failure, with seven redundant copies across seven instances
- Enforce machine roles, ensures loose coupling, and firewalls across machines with separate rolls which arrest cascading failures
- Constantly assess what are the access patterns and network pathways allowing awareness of hotspot in access patterns and are able to adapt partitioning strategies to mitigate them and release stress across the systems
Protect the service with throttling.

- Onus on the service not the user, to provide sensible throttling and resource governance.
- No one service should be able to bring the service (or a significant portion of it) down.
- Resource governance – system should be able to control the minimum and maximum resource allocations, and throttle activities that trending towards exceeding quotas.
3 Challenges Ahead

In order to take the service to the next level in terms of providing ever-improving customer value and also continuing to improve our own business margins, we need to invest along a number of different dimensions.
Execute crisp and granular SLA’s that offer more cost/consistency/performance options.

- A fundamental point of Cloud PaaS is to have customers no longer worry about physical units such as machines disk drives, etc.
- At the same time must it is important for the service provider to play inside reasonably attractive price : performance sweet spots
- Investment opportunity would be to formulate crisper contracts (SLAs) which provide a finer granularity of setting expectations

Solutions:
- Investigate algorithms which aim to jointly optimize choice of query execution plans and storage layout, to meet given SLA at minimum cost
- Consistency Rationing – reduce consistency requirements when possible (i.e. when the penalty cost is low) and raise them when it matters
Hardware Investments.

- Converge on the same hardware SKU as Windows Azure
- Adopt solid state disk technology
- As hardware price points evolve, latch on sweet spots to improve price: performance ratio for the customer, while maintaining comfortable price points for running the service
React rapidly to maxing out capacity and availability of new hardware.

- Balance the time to deploy a new cluster vs. the declining cost of hardware over time
- Anticipate trends in hardware
- Leverage improving hardware efficiencies to benefit the business
System help on performance tuning and Management.

- As underlying infrastructure becomes more stable, main point of failure shifts closer to the customer
- Improve level of self help
- Customer undertakes recommended remedial actions on their own, with existing help and hints from the system
4 Conclusion

Past Lessons

• Design everything for scale-out
• Carefully instrument and study the service in deployment and under load
• Catalog failures and take a calibrated approach to responding to them
• Design the architecture to mitigate failures rather than focusing too heavily on failure avoidance

Ongoing challenges

• Empower the user of the service to get help from the system to figure out failures
• Move towards more granular SLA’s
• Stay on top of hardware trends and algorithmic innovations
• Improve tools both to attract and migrate large and varied customers with mission-critical data and computations to the SQL Azure service