A highly scalable application requires the use of specific patterns and practices. Designing for optimal performance and scale-out is key. Use the patterns below to help you architect your solution and continuously refine your application.

**SCALE OUT WITH SCALE UNITS**
- Use more instances, not bigger hardware. Scale out by deploying additional worker role instances when additional compute is required.
- Use „scale units“ that are a group of role instances and their support resources. For example, a scale unit could be 2 worker roles, 2 monitor roles, 4 queues, and 1 SQL Database instance.

**RECOGNIZED COMMUNICATION**
- Asynchronous communication between the worker and role instances is essential for scalability. Use Azure Service Bus or Azure Service Bus and Message Queue.

**DECOUPLED COMMUNICATION**
- Asynchronous communication protocols ensure that the application continues to function even if one part of the system is unavailable. Use Azure Service Bus to decouple communication between role instances.

**SAVING STATE**
- The durability of service and worker role instances in the cloud can be challenging. Using SQL Database as the primary storage location and a blobstore as a secondary storage location can improve durability.

**NET INT FOR FAULT TOLERANCE**
- Transient issues and errors are unavoidable in large-scale systems. Implement a re-try strategy across the application to provide resiliency against failures. For every retry to quickly resolve an issue, or to report a “soft” failure, ensure that it doesn’t impact service latency and exposure to potential data loss.

**CACHING**
- Use caching to improve application performance by storing frequently used data in a distributed cache. This reduces the load on the database and improves performance.

**FAN-OUT QUERIES**
- Database lookup logic is placed in a cloud service. To find data, multiple databases are queried. Each database query is limited to 3000 databases per query.

**HORIZONTAL PARTITIONING**
- As data moves across the system, use the partitioner to distribute data. The database must be partitioned. Use horizontal partitioning by distributing data across multiple shards.

**VERTICAL AFFINITY**
- When many users access data simultaneously, traffic congestion occurs. Use data affinity to minimize traffic and improve usage. For example, invoices are partitioned by user, stock keeping units (SKUs) and order date to reduce data skew.

**VISIBILITY & MONITORING**
- Visually monitor the application to ensure performance is within acceptable ranges.

**SCALE BIGGER, BETTER, FASTER**
- Enable stability into the app’s lifecycle by increasing performance, reducing response times, and minimizing failures.

**AUTHENTICATION SCRIPT FOR SUCCESS**
- Monitor the application to ensure performance is within acceptable ranges.

**LOAD TESTING: GET TO THE TOP**
- Load test the system with both stress tests and by simulating real-life usage. Vary the load size to determine the optimal scale-out configuration.

Build and Deploy Windows Azure scalable applications to achieve performance, scalability, and costs. You can architect your solution and continuously refine your application.

---

**Plan & Design**
- Build & Deploy
- Run & Tune

**Cloud Applications Using Windows Azure Cloud Services**

© 2013 Microsoft Corporation. All rights reserved. Created by the Windows Azure Team. Designed and developed by Microsoft.com. Field Notes 1.0 (2013)