Scalability

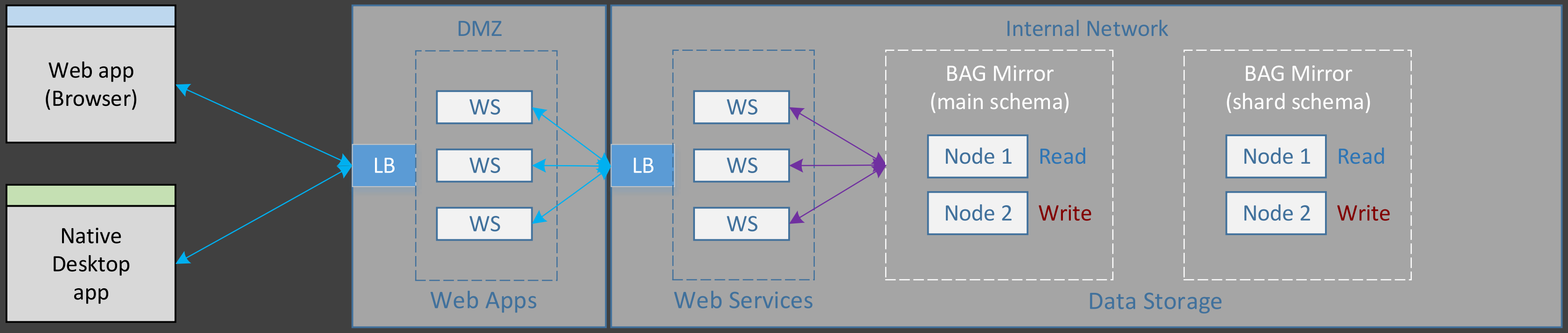
A scalable system offers multiple benefits to an organization. First, it allows each system to start very small, and grow to meet demands for increased processing and storage capacity only if and when it is needed. This helps to significantly reduce the initial expense of setting up a new systems. Second, it allows a system to grow from its small initial deployments to whatever size is needed by the business without requiring any changes to its architecture or code (no need to redesign the system with a new architecture and different programming language to handle large-scale implementations). The upper limits of scalability should be far beyond the needs of most businesses, so that the scalable architecture will never be outgrown.

The scalability of a system is determined by the ability to add various types of resources, and for the system to then be able to use those added resources effectively. Many systems have performance and scalability bottlenecks built into their own architecture that prevent additional resources from being utilized once those bottlenecks are reached. It is important that the mechanisms used for scalability does not interfere with the mechanisms used for availability (redundancy, failover and recovery).

DGP systems in each location are divided into processing nodes (web servers) and storage nodes (database servers). The two types of nodes scale in different ways. Each processing node has its own individual level of performance. The overall processing capacity in a location can be increased by improving the performance of each web server (performance engineering), and/or by increasing the number of web servers in the server farm. The upper limit on the scalability of web server farms will be determined by network bandwidth and the limitations of the load balancers. The load-balanced web server farm is the best and simplest physical topology that provides both excellent horizontal scalability and fault-tolerant redundancy. This is one of the main reasons that so much of the logic and processing workload in a DGP system is consolidated in load-balanced web server farms.

DGP storage nodes scale both vertically and horizontally (as shards). The main database schemas scale vertically by adding more resources to the database servers. DGP's architecture in particular depends on the use of (many) terabytes of very fast solid-state storage (500,000 to millions of IOPS) so that the storage itself never becomes a scalability and/or performance bottleneck. This enables relatively inexpensive servers to meet the needs of very large systems.

Separating databases onto their own dedicated hardware, and partitioning large databases can also be used to increase the scalability of the main database schemas in a location. Beyond that, DGP shards allow data storage to scale out horizontally for most star and snowflake schemas. This technique enables shard database servers to be added to a location incrementally as needed to store growing amounts of data. Lattice FileStore is an example of this type of shard storage.



LB = Load Balancer | WS = Web Server | FS = File Server

Scaling the 4-tier architecture within a location is a matter of first separating each tier onto its own hardware, from which point each tier is able to scale independently of the others. The web server farms handle both N + 1 fault tolerance and incremental horizontal scalability for both the DMZ and internal network. The SQL Server Availability Groups provide redundancy for the various pairs of database servers. The main schema servers can only scale vertically. The shard schema servers provide horizontal scalability for shard data by incrementally adding more shard servers as needed.

Scalability Verification

1. *The horizontal scalability of load-balanced web server farms have been used successfully for decades, and no longer need to prove their capabilities. The processing node scalability of the system as a whole is maximized by consolidating as much logic as possible in the web services of the middle tier.*

2. *Similarly, the vertical scalability of SQL Server and other storage subsystems are tested and verified using their own built-in capabilities as needed.*
3. *The API Tester and AutoWork Tester are able to test and verify that the functionality of the Lattice Filestore hybrid shards is working correctly, and also measure its performance at the same time.*
4. *The scalability of client applications depend on server-side pagination to stay within API response size limits, and integrated search functionality to limit the number of pages of data that meet the search criteria. These features are tested in the API Tester test harness, and also manually in the Lattice UI screens.*
5. *Just-in-time updates of the authorization data cached in each account record by the Login method is the mechanism used to handle the workload of updating potentially millions of user accounts incrementally over time rather than as a massive single process.*

Replication Scalability

The redundancy needed for 100% availability and the scalability of a system intersect in the mechanisms used to replicate data between its multiple locations. The majority of the workload (and potential bottleneck) of data replication falls mainly on the data storage. As mentioned previously, DGP's architecture depends on the use of fast solid state storage with 500,000 or more IOPS to handle the data storage workload of an average system. This level of performance is currently easy and inexpensive to achieve, given commonly available solid state storage performance and prices. Going forward, main memory and storage will eventually merge as some form of persistent RAM, and that hardware is what DGP is really designed for. In the meantime, the architecture works well using fast solid state storage.

Replication is basically an ETL process with no transform, since the source and destination schemas are identical. The extract is a query of a database table implemented as a polling operation. It does not matter if the query of the source is local (push) or remote (pull), since the work of both the extract and load are executed as calls to web service API methods.

The wasted overhead of polling for new records to be replicated is minimal, as long as the data storage IOPS of the database server is not a constrained resource. Therefore, the effectiveness of the merge replication depends on its ability to keep up with the rate at which records are being added to each source table in order to replicate that data to the other destination tables, and vice versa. The AutoWork services that run all of the automated processes in a location are a federated computing system that scales out horizontally to meet growing workloads as the distributed system as a whole scales in size. The scalability of that federated computing system is itself not an issue.

However, the default replication process operates sequentially (finish to start due to the token mechanism), so the frequency of the polling and the batch size of each iteration must be adjusted so to handle the rate at which new records are being created. Each request and response message is limited to 64K, which means the largest batch of records that can be sent or received would be approximately 60K when written as XML. The time required to merge the batch of records into a destination table will determine the minimum iteration interval.

If sequential processing of replication iterations is unable to keep up with the rate at which new records are created, optimistic release of the queue records can be used to handle spikes in the creation of new records. However, if those elevated rates are sustained it could eventually result in too many concurrent threads used by the DGPAutoWork service running on each processing node. The solution is to insure that the AutoWork federated computing system in each location has enough processing nodes to avoid that situation. In addition, every automated process in DGP has been implemented as an API method both for security reasons, but also to insure that it is executed as an IO-bound RPC. These are handled by the .NET Framework as a combination of thread pool threads (managed by the Task Parallel Library) and IO-completion ports, which results in very efficient use of the thread pool threads on each node.

This diagram shows an example of a co-lo rack with separate QA environment hardware.

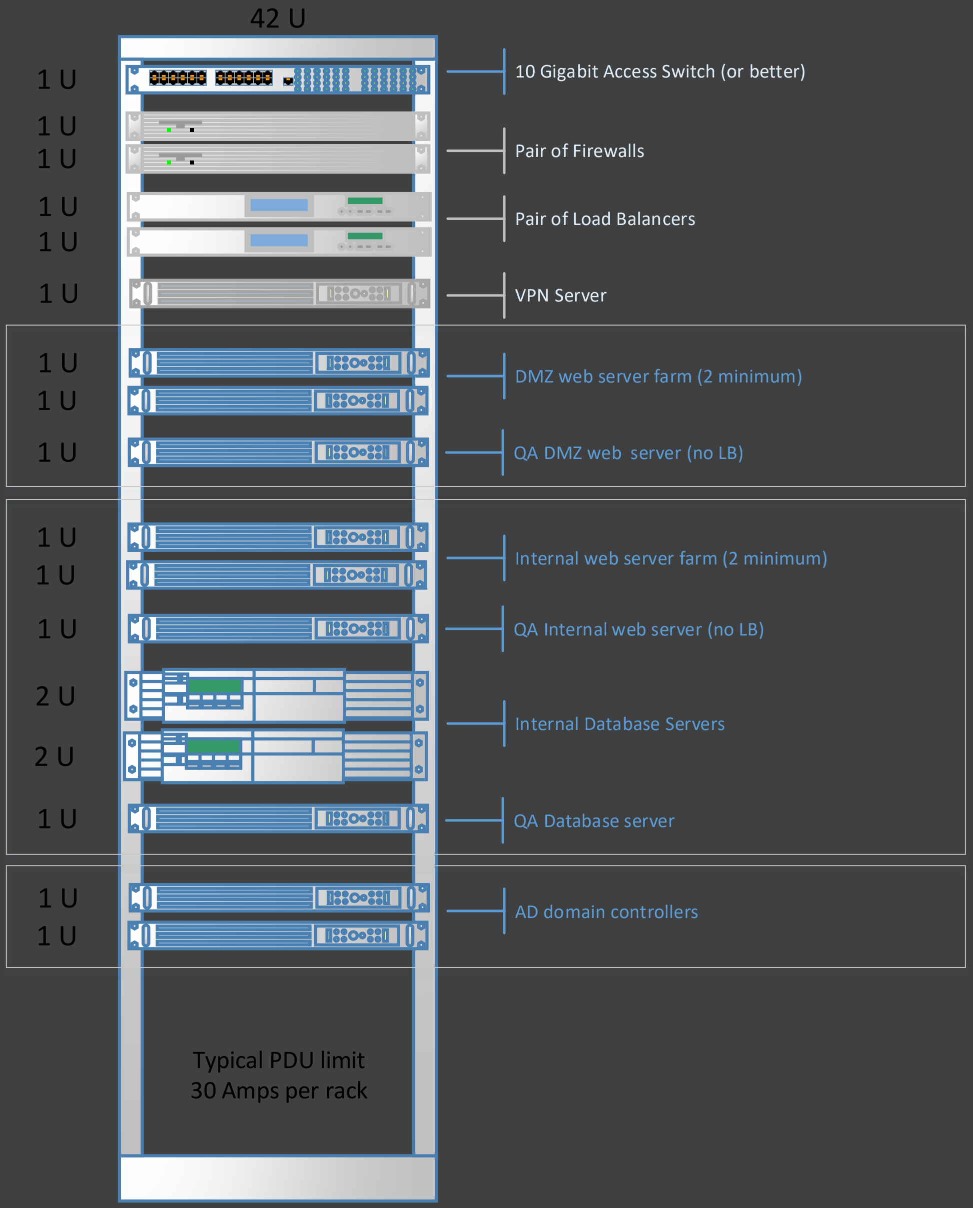
The objective when hosting a system in a co-location rack is for as much traffic between the tiers to remain within the rack as possible (via the access switch).

The access switch should have more bandwidth than required to help avoid over-subscribing the switch fabric.

Firewalls and Load balancers can be hardware appliances or software running on commodity servers.

The database servers must use fast solid state storage (or better), with very low latency and 100,000's to 1,000,000's of IOPS so that the storage tier does not become a bottleneck for the overall system's performance and scalability.

A pair of properly configured database servers can each store 10's of TB of data and support a dozen or more web servers (which for most applications represents hundreds of thousands up to millions of concurrent users).


42 U
1 U
10 Gigabit Access Switch (or better)
1 U
1 U
Pair of Firewalls
1 U
1 U
Pair of Load Balancers
1 U
VPN Server
1 U
1 U
DMZ web server farm (2 minimum)
1 U
QA DMZ web server (no LB)
1 U
1 U
Internal web server farm (2 minimum)
1 U
QA Internal web server (no LB)
2 U
2 U
Internal Database Servers
1 U
QA Database server
1 U
1 U
AD domain controllers
Typical PDU limit
30 Amps per rack