System Scalability

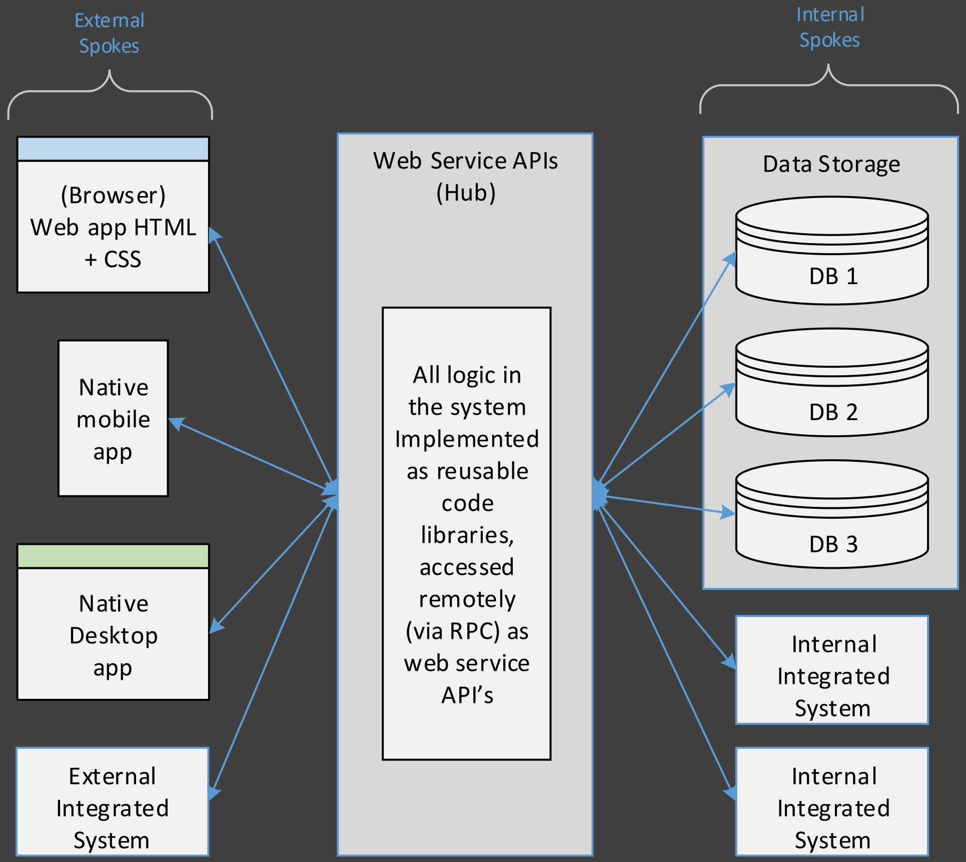
Scalability must be designed into a system from the ground up, just like security, high performance and high availability, etc. As a general rule, almost any type of architecture and implementation can work reasonably well at a small scale, but as the size of a system increases, fewer and fewer architectures are able to continue to work well at larger and larger scales. At the upper ranges of scalability, only a relatively small number of architectures are able to deliver good results. DGP is designed to work very well at a large scale that is beyond the needs of almost all businesses, but at the same time it is also designed to be able to scale down to run well on a single computer. While DGP is intended primarily for the SMB market, it can be useful to organizations of all sizes.

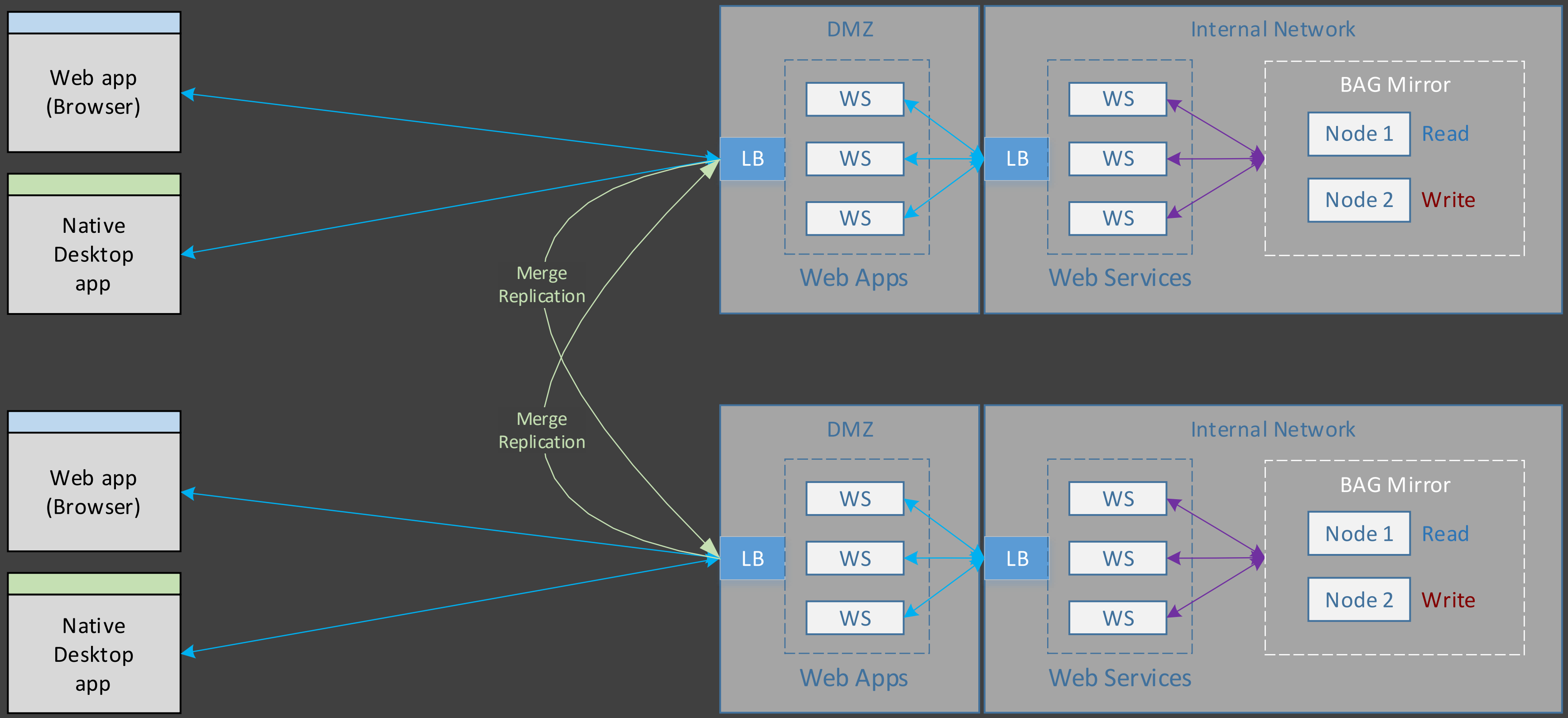
Location Scalability

Scalability in a DGP system occurs within a location, which at a small scale can be implemented as a single computer, and at a larger scale is a distributed modular multi-tier system of many servers connected by a Local Area Network (LAN). A DGP system consists of one or more locations, with 2 locations (a primary and a backup) generally being the optimal number to balance cost vs. 100% system uptime. Scalability of a DGP system occurs within each individual location, while 100% uptime is provided by the redundancy of having multiple locations and by replicating data between them. Optional redundancy within each location can also be used to make each individual location more fault-tolerant as well.

At the scale of a single computer, all the tiers of a DGP system are hosted on that computer and uses the system bus as the "network" (via the localhost loopback address) that allows the tiers to communicate with each other. As a location increases in scale, each tier is given its own dedicated hardware linked together over a Local Area Network (LAN). The different locations are in turn generally linked together over some type of Wide Area Network (WAN) such as the Internet. The scalability of the modular hub-and-spoke system in each location is the combination of the scalability of each major tier and subsystem, strongly influenced by the performance, efficiency and scalability of the software running on each node of the system. DGP systems are generally configured as modular hub-and-spoke client/server systems with 3 logical/physical tiers:

1. Client application spokes (which include the AutoWork Windows services as well as remote integrated systems)
2. Server tier
    a. Web service APIs (the hub of all functionality implemented as a stateless web server farm)
    b. Data storage spokes (database servers, file servers, etc.)


External Spokes
Internal Spokes
(Browser) Web app HTML + CSS
Native mobile app
Native Desktop app
External Integrated System
Web Service APIs (Hub)
All logic in the system Implemented as reusable code libraries, accessed remotely (via RPC) as web service API's
Data Storage
DB 1
DB 2
DB 3
Internal Integrated System
Internal Integrated System



The ability to run all of the tiers of a DGP system on a single computer for small scale systems, and then easily increase the scale of a location by adding more hardware to each physical tier is made possible by the relatively loose coupling between these tiers, with all inter-process communication occurring in the form of various types of RPC's. The only change to a system at different scales is the configuration of the endpoints of the web apps, web services (URLs), data storage (connection strings), and file storage (UNC paths). The scalability of an overall DGP system is best explained by looking at the scalability of each tier of its modular hub-and-spoke architecture, starting with data storage.

DGP Database Scalability

The scalability of most business software systems ultimately depends on the scalability of the data storage tier of a system, since business systems are primarily concerned with the storage and retrieval of data. In the past, the upper limit of both the performance and scalability of a business software system was typically constrained by the IOPS (I/O Operations Per Second) of mechanical disk drives, which had a very small number of IOPS to work with, and software systems had to be designed around those severe limitations. More recently, that limitation has been all but eliminated thanks to solid-state storage.

The scalability of data storage starts with the performance and efficiency of the database schemas and the SQL syntax used for read and write operations. For example, too many indexes can cripple the performance and scalability of writes, while not enough indexes can cripple the performance and scalability of reads. Poorly written and un-optimized SQL syntax in general will decrease both performance and scalability. Beyond that, each database server can be scaled up vertically by adding more resources such as CPU cores, RAM and solid-state storage to a server. Partitioning databases onto their own dedicated hardware is the final level of data storage scalability, with database shards used as an additional level of partitioning for some schemas (DGPDrive as one example).

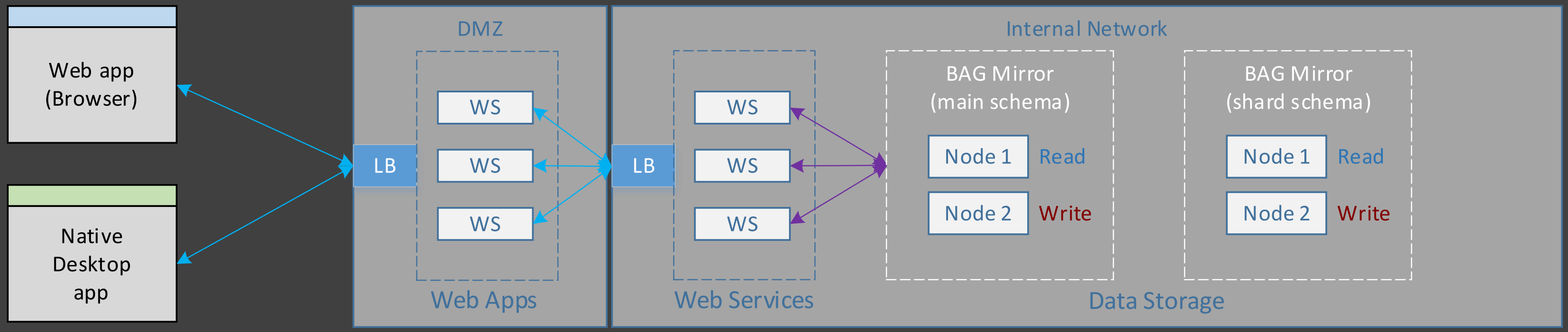
In terms of vertical scalability, it is relatively easy and inexpensive for each server to have many TB of solid-state storage with GB/sec throughput and millions of IOPS. What this means in practice is that the data storage IOPS is no longer the main scalability bottleneck, which in turn enables polling operations to be a viable alternative to event driven push mechanisms. These types of polling processes are extremely important to the guaranteed “once and only once” idempotent operations that various parts of the DGP architecture depends on. This is why the DGP architecture was really not a viable alternative until solid state storage eliminated the mechanical disk I/O bottleneck and eventually became inexpensive enough to be used for most software systems.

The scalability of each database server is a combination of:

- The number of transactions per second a server can process
    - Storage IOPS
    - The number of CPU cores and amount of RAM

  - The performance, efficiency and scalability of the database engine used
  - The database schema (number of indexes, constraints, etc.)
  - The performance and efficiency of the SQL syntax (DML create, read, update, delete, etc.)
- The volume of data that can be stored on and retrieved from each server
  - WAN bandwidth (LAN bandwidth should not be an issue)
  - Data throughput (GB/sec) for data written to and read from a database server
  - The amount of solid-state storage (TB) of each server

The overall scalability of a system is further increased by using partitioning to separate each major database onto its own dedicated database server hardware, and then vertically scaling up each of the database servers as needed. Some database schemas can also use database shards (such as DGPDrive, for example) as another level of partitioning that provides a degree of horizontal data scalability, adding more shard partitions to a system as needed.



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

Web Server Scalability

The scalability of the web service APIs that act as the hub in the hub-and-spoke architecture starts with the performance and efficiency of the web services themselves. The duration of the server-side processing of each API method call will determine the number of requests per second that can be processed by each CPU core.

While the web server farms in a DGP location allow more web servers to be added to a location (horizontal scalability), the performance and efficiency (and resulting scalability) of the web service APIs running on each server is also important. The high performance and efficiency of the web services themselves can help to improve the scalability of each web server in a location, reducing the need to add more servers to a server farm, which helps to reduce the total cost of a system.

The scalability of a typical stateless web server farm is a combination of:

- WAN bandwidth (LAN bandwidth should not be an issue)
- Data storage performance and scalability
- The number of HTTP requests per second that each server is able to process
    - The time it takes to process each request on the server will ultimately determine the number of requests per second that can be processed by each core
    - The number of CPU cores in each web server, which determines the size of the thread pools on the server
    - I/O completion ports which improve the utilization of thread pool threads for I/O-bound RPC's, such as calls to a database server which are a majority of the work done by each API method
- The number of web servers in the server farm
    - The capacity of the load balancer cluster
        - Whether TLS is terminated on the load balancer or on each individual web server
        - Whether or not any of the web servers require session affinity (sticky sessions)
    - The capacity of the location infrastructure to add more web servers

In a DGP system, 80 to 90% of the API methods make a call to one of the database servers in a location using direct ADO.NET connections. This means that each of those worker threads managed by IIS is executing an I/O-bound RPC to a database server. In Windows, these types of calls are automatically handled using I/O completion ports, which release the worker threads back to the thread pool for the duration of the asynchronous network activity. For this reason, using synchronous calls for I/O-bound RPC's will generally be the best and simplest solution overall (no need for async calls in the web service logic due to the I/O completion ports).

Also, it is best to avoid mixing the use of OS-level threads and local .NET framework threads within web service API methods. Nesting worker threads within IIS threads can lead to thread starvation issues. This is compounded by the fact that OS-level IIS threads and .NET local threads are generally unaware of each other, which can lead to unusual behavior that is very difficult to debug.

Security System Scalability

DGP web services are stateless message-based APIs that have the functionality of an API security gateway integrated into each web service in the form of a reusable library (the message processing pipeline). When caching is turned off, each API call is individually authenticated and authorized, which requires a query to the identity store database. The identity store database has been designed so that authentication queries select a single record from a single well-indexed database table, with the returned record containing all of the authorization data for the authenticated account. This requires that a copy of the authorization data for all API methods be cached in each account record, along with a similar authorization read list and write list of the data security groups. This cached data is updated per account, on demand whenever a user calls the login method when connecting to a location.

The scalability (and performance) of this data-driven Role Based Access Control (RBAC) identity store is improved by:

- Caching of the UserInfo object on each web server, which contains all the method and data ACL's for the account, eliminating the need to query the identity store database (cache aside pattern with a relatively short expiration, using the ASP.NET cache).
- Caching the ACL data for API methods, data group read list and write list into each user account database record, so that the authentication query is a search for a single record from a single table (no joins or complex queries needed).

- The user account table is indexed and optimized for fast reads.
- Just in time lazy updates of the ACL data cached in each user account occurs each time a user calls the Login method. This mechanism to update the authorization data summary in each record breaks up the huge authorization update workload into small discrete tasks executed individually, as needed.

AutoWork Scalability

DGP contains an optional automation subsystem designed for high performance, high scalability, and excellent reliability. Its primary purpose is to run the data replication and data verification processes, but it is a general purpose automation subsystem that can be extended to run almost any type of automated processes just by adding more automated process API methods (system evolution).

All logic for the AutoWork subsystem has been implemented as API methods, as are all of the automated processes themselves. Process orchestrations (a series of sequential steps) are actually implemented as methods of reusable code libraries, which are either called directly in other methods or are called remotely using the API methods.

In general, the AutoWork Windows service acts as a scheduler, with timer events triggering calls to one-way process methods in different locations. Some of the API methods query various database tables used as queues of work waiting to be executed, while other API methods orchestrate the steps of the automated processes themselves. Internal process orchestrations that run entirely within a single location are able to call internal libraries directly, executing in the memory space of the worker thread itself, which is many times faster than an RPC to a remote web service API method.

Client App Scalability

The scalability of client applications refers to the ability of the app to always provide a good user experience and good performance no matter how much data is stored in a system. The “scalability” of client applications is primarily a matter of server-side pagination integrated with all search methods to limit the maximum number of records returned to a client app. The pagination must be integrated with the flexible search criteria when creating data result sets to be displayed by the client applications.

There are several types of server-side pagination. By default, DGPDrive uses the SQL Server database engine, and its Order By/Offset/Fetch syntax to paginate query data. The performance of this syntax depends on having good cover indexes for each table, and generally works very well for tables with up to several million records (depending on the database server hardware). The Search functionality accepts various input parameters to build customized SQL Select syntax that are then integrated with the server-side pagination syntax. For tables with very large numbers of records, an indexed placeholder query mechanism is available as a more efficient, but also slightly more complex server-side pagination alternative.