DGP Automated Processing Overview

The objectives for DGP's automation subsystem are to create an easily extensible, data driven, high performance, highly scalable and fault tolerant means of executing automated processes with the same strong security as all of DGP's other API functionality. To reiterate some of the requirements that affect the design of the subsystem:

- Break up large processing workloads into continuous, incremental iterations of small batches of work. For complex processes, this requires:
  - Some sort of placeholder value that can be used to track the progress of the processing work as it incrementally crawls through the large backlogs of data to be processed.
  - The state of the progress and results of each iteration must be stored, updated and reused from one iteration to the next in a fault-tolerant, reentrant manner.
  - Error handling and logging must themselves be global and fault tolerant in order to provide a good level of observability into the execution of the many automated process iterations.
- Allow the iterations of batch work to be scheduled in a variety of ways (continuous intervals, fixed times, etc.)
- Decouple the process to schedule iterations of work and trigger their execution at the scheduled time from the process of performing each batch of work itself. Problems must not affect or delay the automation schedule.

In order to meet all of these requirements, DGP's automated processing subsystem is designed so that every process orchestration is implemented as a one-way web service API method. This allows the web servers in the stateless server farms to handle the multi-threading of the automated processing workload, and also ensures that all such workloads are implemented as IO-bound RPC's (the calls to web service API methods, calls to database engines, etc.). The SOAP one-way "fire-and-forget" request/acknowledge pattern decouples the scheduled trigger of the execution of an iteration from the subsequent background execution of the batch of work itself.

Windows has a feature called IO Completion Ports that are used to efficiently manage the thread pool threads used for such processes. DGP is designed to take full advantage of that efficiency. Large workloads are broken down into small batches of work, performed incrementally as continuously running background processes. This includes all of the work of the automation subsystem itself, which have also been implemented as API methods.

By using DGP's message-based APIs, the automation API methods are run with the full API security in terms of authentication, authorization and encryption, etc. Processes which need to maintain state info have their own work tables acting as a queue to schedule recurring iterations of each process and also to maintain the ongoing state values used by the process between iterations. DGP initially has three types of processes:

1) Data replication, which has its own queue table to maintain the incremental replication state per database table.
2) Data verification, which has its own queue table to maintain the incremental verification state per database table.
3) General processes, which has a queue table for processes that have no state or very simple state which can be documented as structured text from one iteration to the next (and therefore do not require their own dedicated queue tables).

In this context, the DGP AutoWork Windows service is basically a simple scheduler, using timers to query (polling) for iterations of work that are ready to be executed. The Windows service contains logic to claim records from a queue table that are scheduled for execution, and then pass the queue records into one-way "fire-and-forget" API methods that execute each iteration of a given process, handle the logging of information and/or errors, update the state data of the queue record and schedule the execution of the next iteration. Each web server in the stateless web server farm manages the multiple threads of the fire-and-forget processes running in parallel using Windows IO completion ports. The number of servers in the load-balanced web server farm are adjusted to easily handle the combined workload of normal API method calls and the automated processing API method calls in each location of a production environment.

DGP Automated Processing Performance and Scalability

AutoWork Scheduler

The steps of the scheduler process include:

1. Claim a small batch of queue table records
2. For each record claimed, call its corresponding one-way "fire-and-forget" SOAP API method

The scheduler contains a timer for each type of automated work. Each timer event handler executes the logic to claim a small batch of records from its respective queue table. For each record claimed, the appropriate one-way API method is called. The rule for this

part of the automation process is that all of the one-way API methods in a claimed batch must be able to be called in less time than the timer interval. The one-way methods avoid the blocking of a synchronous method call by using a request/acknowledge mechanism that immediately returns an acknowledgement after receiving the request, while the worker thread on the server continues to execute the one-way API method in the background. Therefore, network latency should be the only major factor determining how long it takes for each timer event handler to sequentially call all of the one-way API methods in the batch of claimed queue records. The batch sizes and timer intervals are configurable to ensure the timer interval work does not overlap.

The work (queue) tables serve multiple purposes:

- Schedule the time to run each process iteration using various schedule rules (iteration time interval, recurring fixed times, etc.)
- Maintain process iteration state data, which is only relevant within each location and is therefore not replicated.
- Claiming queue records acts as a token granting the exclusive right to execute an iteration of a process.

Each API request and response message is limited to 80K in size in order to avoid instantiating objects in the .NET Framework Large Object Heap – (anything over 85,000 bytes or approx. 83K). This results in the processing of small batches of records incrementally and maintaining ongoing state information about the process in the queue tables from one iteration to the next.

Each queue record must be updated with a unique ID (GUID) when it is claimed by an AutoWork scheduler to differentiate the work it does from all the other iterations. The use of SOAP one-way "fire and forget" API methods allows the automation workload to scale as part of the same stateless web server farm used for the regular API methods, while also taking full advantage of the Windows I/O completion ports for their efficient use of the thread pool threads on each web server.

One-way API Methods

The steps of the one-way API method for data replication include:

1. Accept the input values from the replication queue record
2. Call the source API method polling for source records to be replicated (synchronous)
3. If any source records are found, pass the table of source records into the destination replication API method (synchronous)
4. Calculate the time for the next scheduled iteration of the source table replication process
5. Update the queue record with the results of the process steps and next scheduled iteration time

The steps of the one-way API method for data verification include:

1. Accept the input values from the verification queue record
2. Call the source API method polling for source records to be verified (synchronous)
3. Pass the table of source records into the destination verification API method (synchronous)
4. Calculate the time for the next scheduled iteration of the source table verification process
5. Update the queue record with the results of the process steps and next scheduled iteration time

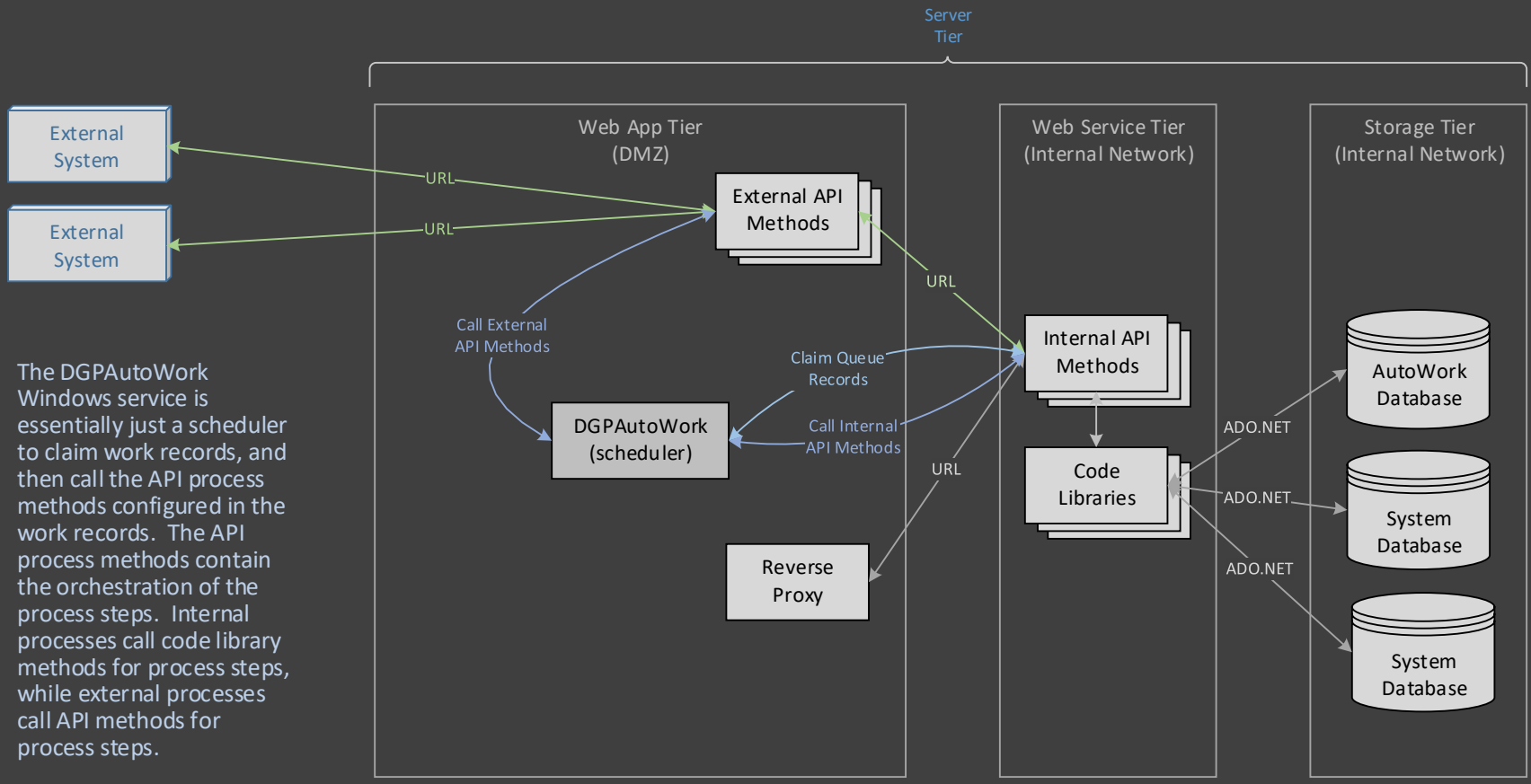
The steps of the one-way API method for general processes include:

1. Accept the input values from the general process queue record
2. Call the API method indicated by the general process record
3. Calculate the time for the next scheduled iteration of the general process
4. Update the queue record with the results of the process steps and next scheduled iteration time

The replication and verification processes have enough state data to require their own queue tables to manage it all from one iteration to the next. The general table is used for processes which have no state data to be managed, or a small amount of simple state data that can be represented as a delimited string of values.

The current implementation of DGPDrive is based on reusable queue table records combined with a second automated process to periodically scan the records of the queue tables themselves to detect any problems. Errors are logged to both the local Event Viewer and to a centralized database. Detailed information about each iteration of the automated processes is written to log files (segmented per time interval to prevent the log files from becoming too large).

DGP process methods are implemented as SOAP One-Way methods and follow a “request/acknowledge” pattern that immediately returns an acknowledgement of the receipt of the request instead of returning a response. The HTTP request is released as soon as the acknowledgement is returned to the client app. Only the SOAP protocol provides the request/acknowledge one-way functionality. The use of one-way methods allows a single AutoWork Windows service to trigger the execution of a large volume of automated process iterations, since the scheduler is not blocked waiting for each process iteration to be completed. Multiple AutoWork services can be run as needed for fault tolerance and to keep up with the automation workload. The database engine easily handles the contention of the (deliberate) race condition when claiming work records.



The DGPAutoWork Windows service is essentially just a scheduler to claim work records, and then call the API process methods configured in the work records. The API process methods contain the orchestration of the process steps. Internal processes call code library methods for process steps, while external processes call API methods for process steps.

Each automation process method runs as a one-way "fire and forget" (SOAP one-way request/acknowledge) API method. The API methods are called by the DGPAutoSvc using the credentials of a system account stored in an encrypted section of the Windows service app.config file. The orchestration of process steps is built into the API process methods. Each automated process method performs its own logging, update of the claimed work record and scheduling of the next iteration, etc..

Some automated processing is run outside of the internal network to enable the processes to call out to external systems, which is not allowed from within the internal network (if a DMZ is present). Scalability of the automated work is handled by the horizontal scalability of the web servers in the web server farms. Multiple scheduler services can be run both in the DMZ and the internal network, but since they only call one way methods, and do none of the actual automated work themselves, the scalability of those windows services is generally not an issue.

One-way API Method Scalability

The one-way methods allow for the parallel execution of different automated process iterations to be managed by the web servers of the load balanced web server farm. The one-way request/acknowledge pattern provided by the SOAP protocol produces the best results for large scale automated processing. Alternative prototypes of parallel execution (HTTP services, Windows services using .NET thread pools) were not able to produce good results at scale.

By default, the iterations of an individual process are limited to sequential finish-to-start execution of subsequent iterations due to the claimed queue records behaving as an authorization token. This mechanism acts as a throttle for each automated process, preventing new iterations of a process to be scheduled until the previous iteration has been successfully completed. The frequency of the iterations and the number of records processed as a batch can be adjusted to keep up with the rate at which records are being created in each database table of a system.

Beyond this, an optional additional level of parallel execution can be achieved by the optimistic release of the authorization token (the work queue record) ***before*** the work of an iteration has been completed. This assumes that an automated process itself has become reliable enough so that the optimistic release of the token before the work has been completed would rarely cause any problems. By default, this option is currently not used, and a mechanism to limit the number of parallel threads each process would be needed.

Queue Tables: all of the automated processes could be run using the General schedule table, but the replication and verification processes have a significant amount of configuration and state information to be maintained for each iteration. This info could be maintained using structured text (JSON or XML) in the General table, but this technique becomes fragile when process iterations experience problems. Maintaining the structure of the state information as a database table schema is much more resilient, which is why the complex processes each have their own queue tables. All other processes use the General table and maintain their minimal state information as structured text. Regardless, the process state info can be recovered fairly easily if it is lost or corrupted.

Claimed Token: the queue tables are designed for the scheduler instances to race to claim a small set of work records in each queue table. The database engine handles the contention of this race to claim the queue records. Afterwards, the records claimed by a specific scheduler act as a token that grants the exclusive “right” to execute the process by whichever thread owns it. A monitor process periodically scans the queue tables looking for problems in the queue records (mainly “stuck” iterations that are taking too long to complete), resets those records, and logs the problems.

Once and Only Once: The objective for all automated work processing is to guarantee that each iteration of work is executed "once and only once". In practice, this is accomplished by using an "at least once" persistent polling process to run an "only once" idempotent work process. In order to execute all automated processes under the full protection of the existing layers of DGP API security, all of the automated process logic has been implemented as web service API methods. The process methods are generally an orchestration of the calls to the library or API methods that represent the individual steps of each process.

Fire and Forget: Each queue record claimed in a polling iteration is executed as a self-contained "fire and forget" API method. Each process is responsible for updating the claimed queue record with results when an iteration of work has been completed.

Dynamic Scheduling: Part of the process to update to the claimed queue record also schedules the next iteration of that task. If the number of data records processed in the prior iteration equals the maximum batch size, the task will be scheduled for immediate execution in order to work through backlogs of records as quickly as possible. Otherwise, the next iteration of a task is scheduled based on the configured interval value.

Sequential Iterations: Each iteration of an automated process is executed sequentially by default (enforced by each queue record acting as a token of ownership). As long as the number of iterations per second and the number of records processed in each iteration batch is able to keep up with the rate at which new records are being created, this default configuration works well for most systems. The batch size and interval duration can be adjusted to help keep up with large processing workloads, while the optimistic release of the queue records can also be used if sequential iterations are not able to keep up with the workload.

Automated Processing Workload

The polling interval for the queue tables is configurable, and depends on how many AutoWork scheduler Windows services will share the scheduler workload. For example, on a single server, a timer interval of 200 MS will result in an average of 5 queries per second of each queue table. Each query attempts to claim a small batch of records, the maximum size of which is also configurable. The rule for setting this polling interval and batch size is that the timer event handler must be able to finish calling the one-way API method for each claimed record in less time than the timer event interval. The network latency of the one-way API method calls should be the main limiting factor, since the acknowledgement usually is returned very quickly.

The overall DGP architecture has been designed to depend on various types of polling, which was *not a viable option* in practice prior to the availability of inexpensive solid-state storage which removed storage IOPS as the primary constrained resource. Therefore, DGP depends on the use solid-state storage with 500,000 IOPS at a minimum. It is not difficult or expensive for a server to use solid state storage with over 1 million IOPS. With a 200 MS polling interval, the total database polling overhead for all of the queue tables would be 15 queries per second for a single server installation (3 queue tables queried 5 times per second).

The polling of the replica source tables in each location results in a theoretical maximum of 50 queries per second for DGPDrive (10 replica tables queried 5 times per second). In practice, the actual number of source table queries is less than that maximum due to the lag between the time at which records become available and the time they are claimed by the scheduler. The theoretical maximum overhead for polling in a DGP system (queue table polling plus source table polling) is approximately 65 queries per second, run on a database server that should have at least 500,000 IOPS. In this context, the system overhead for these polling operations is insignificant. Some percentage of those queries will find a batch of automated work to be done, while the rest return no results.

When multiple instances of the DGPAutoWork Windows service are being used, the SysWork database easily handles the contention of the race to claim queue records. This deliberate race condition provides both high performance and a very fault tolerant mechanism to ensure that each automated work iteration is executed soon after it becomes available. One of the objectives of the automation subsystem is to reduce the lag between a work record becoming available and the time it is claimed for execution.

The claimed record mechanism forces sequential execution of iterations (which is a requirement for many automated processes in any event). Optimistic release and rescheduling of work records before the work has been done allows a degree of overlapping parallel execution of the iterations of each automated process itself, and can be viable as long as the processes being optimistically released are themselves very reliable.

The parallel execution of different one-way processes (usually one per source table for both replication and verification) is handled by the load balanced web servers in the internal web server farm. The orchestration of the steps of these processes occurs within each one-way API method worker thread. Some of the steps of these processes call other web services as an IO-bound RPC. Windows IO completion ports help to efficiently manage the thread pool threads on each web server.