WebSphere® DataPower architecture often includes integration with multiple endpoints. Downstream services may be front ended by DataPower for protocol transformation or authorization, orchestrations may be performed within a DataPower service's policy rule, and logging services are often accessed. DataPower count and duration monitors and service level management configurations are well suited to mitigate performance variances within these endpoints. This article describes some of the methods for analyzing the resources allocated to these interactions. The article also describes how you can use DataPower service management tools within policy rule connections in addition to downstream services.

Introduction

IBM® WebSphere DataPower Appliances (hereafter called DataPower) are purpose built for the rapid deployment of system integration and security policies. Firmware and hardware components are matched for optimized policy execution in a hardened and easily managed platform. DataPower accelerates the time to value and lowers the total cost of ownership of these complex infrastructures.

DataPower configurations often implement solutions through integration with other services. For example, security policy decisions may be made by accessing a centralized directory (LDAP). Enterprise policies may be obtained through registry and repository systems. Logging may be performed by accessing a SYSLOG resource. Messages may be transformed or "enriched" through database access. Functions such as these are typically performed before the message is delivered to a downstream application for processing and there may be additional processing of the application's response before the response is delivered back to the client.

This centralized topography demonstrates DataPower’s agile connectivity capabilities. However, each of these integrations may impose latencies or limitations on the success of a process flow. While some interactions may be asynchronous or "fire and forget", others will be synchronous and require completion before subsequent actions can begin. In both instances, transactions may queue up and consume appliance resources waiting for events to complete. While DataPower hardware platforms provide faster interfaces, extended memory, and faster CPUs, in extreme situations these events may limit the ability to process transactions at an optimum rate.
There are best practices to manage these interactions. Monitors may be constructed to track the rate of incoming transactions and the duration required for each to process. Transactions may be "shaped" and processed at predetermined rates. Service level agreements and service level monitors may be configured with complex mediation capabilities, which may be coupled with enterprise governance resources such as WSDLs, WS-Mediation, and WebSphere Registry and Repository (WSRR).

This article discusses and demonstrates techniques to make your DataPower configurations more resilient to the variances of integration dependencies, providing a more robust DataPower architecture. We'll describe some of the fundamentals of DataPower resources across platforms and firmware revisions and review the basics of resource monitoring. We'll then look at some best practices you can implement to optimize your DataPower services.

**Overview of DataPower resources**

DataPower undergoes continuous improvement in firmware and hardware design. DataPower is designed with multiple form factors including 1U, 2U, Blade, and Virtual editions. There are differences in the physical characteristics within these appliances. We'll describe some of those differences and some fundamentals of resource analysis.

**Device capacity issues**

The current DataPower hardware 71XX platforms (7198 and 7199) offer both 1U and 2U appliances. Previous generations, including the 9235 (9004), consisted of a 1U model. While many of the topics that are described in this article relate to all DataPower hardware platforms (including the Virtual editions), we will focus on the current generation, the 71XX/4195 platforms.

The 7198, 7199, and 4195 Blade models vary in the amount of physical memory and hard disk drive space as shown in Table 1.

**Table 1. Model 7198 and 7199 memory and HDD sizes**

<table>
<thead>
<tr>
<th>DataPower model</th>
<th>Hard drive array</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>7198-32X XG45: 1U</td>
<td>Two 300 GB HDD</td>
<td>24 GB</td>
</tr>
<tr>
<td>7199-42X XI52: 2U</td>
<td>Four 600 GB HDD</td>
<td>96 GB</td>
</tr>
<tr>
<td>7199-62X XB62: 2U</td>
<td>Four 600 GB HDD</td>
<td>96 GB</td>
</tr>
<tr>
<td>4195-XXX XI50B: Blade</td>
<td>Two 300 GB HDD</td>
<td>12 GB</td>
</tr>
</tbody>
</table>

**Firmware capacity issues**

The recent DataPower firmware release (5.X) provides enhanced processing capabilities with extended memory support, allowing larger message processing on XG45, XI52, XI50B, and XB62 as well as supporting higher concurrency (see Tech note: Total amount of memory of DataPower 7198/9 devices). These latest models are required to fully utilize the extended memory capabilities of 5.X. You can use 5.X on the previous generation hardware, such as the XI50, XS40 and XA35 model 9235 appliances. You can run firmware versions prior to 5.X on the XG45, XI52, XB62, and XI50B, but you will only fully utilize the increased capabilities with 5.X on XG45, XI52, XI50B, and XB62.
Resource monitoring basics

DataPower provides many "status providers" (or monitoring agents) built within the DataPower firmware for the fetching of status data. These providers are used to determine the health of components, such as fans, temperature sensors, physical memory, CPU, and so forth. For additional monitoring information, best practices, and examples, refer to Monitoring WebSphere DataPower SOA Appliances.

While several status providers are indicators of system resource utilization, memory is a good indicator of transaction efficiency and we'll review some of the specific memory status information here. Status data can be fetched from the WebGUI (as seen in Figure 1) or by using the Command Line Interface (CLI), the XML Management Interface (XMI), or by polling through the System Network Management Protocol (SNMP) Management Information Block (MIB). Each technique will fetch the provider's current status information.

Figure 1. WebGUI memory status

Memory categories

The memory categories shown in the Memory Usage report are confusing at first glance. The primary issues revolved around the total amount of physical memory on the appliance (installed memory) versus the amount available to the DataPower firmware (total memory). When DataPower requests a block of memory from its operating system and completes the requirement for its use, it is typically returned to the hold queue, not to the operating system. It is only returned to the operating system during periods of memory constraint, or when the system recycles. Therefore, the requested memory typically stays even or grows over time. Table 2 shows the memory categories.

<table>
<thead>
<tr>
<th>Category</th>
<th>Amount</th>
<th>Calculation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory usage</td>
<td>3</td>
<td>%</td>
<td>Percentage of memory that is in use.</td>
</tr>
<tr>
<td>Total memory</td>
<td>82333842</td>
<td>kilobytes</td>
<td>The amount of installed memory minus the amount of reserved memory.</td>
</tr>
<tr>
<td>Used Memory</td>
<td>3230980</td>
<td>kilobytes</td>
<td></td>
</tr>
<tr>
<td>Free Memory</td>
<td>7512862</td>
<td>kilobytes</td>
<td></td>
</tr>
<tr>
<td>Requested Memory</td>
<td>355884</td>
<td>kilobytes</td>
<td></td>
</tr>
<tr>
<td>Hold Memory</td>
<td>627904</td>
<td>kilobytes</td>
<td></td>
</tr>
<tr>
<td>Reserved Memory</td>
<td>18853358</td>
<td>kilobytes</td>
<td></td>
</tr>
<tr>
<td>Installed Memory</td>
<td>99197400</td>
<td>kilobytes</td>
<td></td>
</tr>
</tbody>
</table>
Used memory | 3230980 | Total - Free | The amount of total memory minus the amount of free memory. The used memory does not include any hold memory.
Free memory | 79102862 | | The amount of memory that is not in use. This memory is, therefore, available. The free memory value includes hold memory that is not currently in use.
Requested memory | 3858884 | | The amount of requested memory. The requested memory is not reported as used memory until the memory is actually in use.
Hold memory | 627904 | | The amount of memory that is pre-allocated by the appliance.
Reserved memory | 16863558 | Installed - Total | The amount of installed memory minus the amount of total memory.
Installed memory | 99197400 | | The amount of physical memory in the appliance.

**System usage**

Another classification of DataPower resources is available through the "Show Load" command of the Command Line Interface or the "System Usage" (see Figure 2) status provider. The System Usage shows data for several tasks running on the appliance, not just the main DataPower task. The values are displayed as percentages over an "interval", which may be modified through the "load-interval" command. Other tasks include DB2®, SSH, and potentially other tasks that run as side processes and not within the DataPower address space.

**Figure 2. System usage status**

![System Usage](image)

The system usage takes into account all the resources that have been allocated, regardless of whether it is being actively used or simply held in reserve. These values are sometimes useful when working with DataPower support on resource issues. However, the memory usage from the show memory status provider is a more accurate measure to use for capacity planning because the hold memory is available to DataPower for re-use.

**Analyzing memory usage**

DataPower implements transactional processing through services, such as the Multi-Protocol Gateway (MPGW) or Web-Service Proxy (WSP). Services are typically configured within domains
for ease of life cycle management and other administrative benefits. Processing policies are containers that implement rules and the rules contain actions. Actions implement higher level functions, such as digital signatures, encryption, and authentication, or custom processing, through the execution of the Extensible Stylesheet Language (XSL) transformations. Let’s review some of the methods for determining memory requirements for these services.

**Domain memory usage**

The memory information has been enhanced over recent releases to show incremental utilization by domain and service and includes XSL and XML document caches. Refer to the Information Center for complete memory status information for your particular firmware release. Figure 3 shows an example of domain memory utilization. Notice that the display includes values for time increments, the service lifetime (since the last restart), and the document and stylesheet caches. If you are interested in determining areas of your configuration that may be accountable for excessive memory usage, then a good place to start is to look at the domain memory statistics.

**Figure 3. Domain memory utilization**

![Domains Memory Usage](image)

**Service memory usage**

Having identified a domain of interest, you will then want to understand the services within the domain and how they are utilizing memory. From either the default domain or from within an application domain, you can show the specific services and their memory usage. Figure 4 shows an example of the service status information.

**Figure 4. Service memory usage**

![Service Memory Usage](image)

Since the publication of the previously mentioned developersWorks article (DataPower release 3.8.2), additional memory status information has been added. In particular, a new log category "memory-report" now produces detailed information about each individual action’s memory utilization within a processing rule. For example (as shown in Figure 5), a sample rule execution demonstrates the ability to determine the memory used by sign, verify, and transform actions and the transaction in total. This is particularly valuable in custom XSLT actions. XSLT that uses inefficient XPath or patterns may often be optimized to reduce memory footprints. The report shows memory information for the initial parsing and associated schema validation of incoming messages and each action with the rule. The sign action in this particular transaction is using more memory resources than the simple identity transformations that precede it, as you would expect given its complexity.
Service implications on memory

There are multiple factors that affect memory utilization. Message sizes and concurrency are obvious factors. As transactions are processed, DataPower flow rates are affected not just by the "work" that DataPower applies, but also by the interactions with "off box" resources. Logging steps, for example, may be dependent on the success or failure of the logging resource. Application resources may ultimately have to process the transaction and the response from that application may need to be further processed. In this section, we'll discuss some of the memory utilization factors in more detail.

Size of input messages

With the use of firmware version 5.0 and hardware model 719x platforms, very large messages may be processed. While every environment has uniqueness and every message varies in complexity and structure, processing messages of many gigabits is possible, including complex operations such as digital signature and encryption processing. One factor to consider when processing XML or SOAP messages and when using actions within a policy that processes those documents is the required "parsing". Parsing, or the processing of an input byte stream into a dynamically accessible object structure, requires memory that is significantly greater than the input stream itself. This resource requirement is multiplied in cases of concurrency. We'll see shortly that it is possible to "stream" messages without fully parsing, allowing for the effective processing of unlimited sized messages.

Asynchronous and synchronous actions

DataPower actions may be executed as "synchronous" in which subsequent actions wait for completion, or "asynchronous" in which actions run in parallel. By default, actions are synchronous, each waiting for its preceding sibling to complete. Normally, this is the desired behavior. Certain actions, such as authentication and authorization (AAA) or service level monitoring (SLM) should only be run synchronously as subsequent actions are executed based on
their successful execution. However, for some policy rules, it is possible to run actions in parallel. An example is posting log data to an external service. If the log server slows down, you may not want the client's transaction to delay if the log event is non-critical.

However, asynchronous actions are not cost-free. DataPower is primarily optimized for minimizing delay. As a transaction executes each action in a rule, it does not free the memory used until after the transaction completes. Rather, it puts that memory in a "transactional or hold" cache for use by subsequent actions. The memory will only be free after the entire transaction has completed. It is not available for use by another transaction until such time.

Asynchronous actions can overuse resources in conditions where integrated services are slow to respond. Consider an action that sends a SOAP message to an external server. The result of this action is not part of transaction flow and you do not want to delay the response to the client waiting for confirmation from the server. The action can be marked asynchronous. Assume that normally the external server responds with a HTTP response after just 10 milliseconds (ms).

Now assume that you have a modest 100 transaction per second (TPS) flow to the device and that the external log server has a slowdown and does not respond for 10 seconds to each SOAP message. Assume each transaction uses 1MB of memory, parsing and processing the request transaction. Suddenly, your log actions are holding 1GB of memory as they wait for the HTTP responses from the logging server! This can quickly cause the device to start delaying valuable traffic to prevent over use of resources. If this logging is not business critical, you might want the logging actions to abort before the main data traffic is affected. We'll describe how that is implemented using a controlling or "façade" service, described in Implementing a service level management.

Streaming

An alternative to document parsing is the "streaming" of documents through a service policy. In this scenario, the document passes through a policy rule, section by section, and while the entire document is not accessible, this is often all that is required. In streaming mode, memory requirements are greatly reduced. Streaming requires strict adherence to processing limitations, including XSLT instructions that may be invoked. For example, an XSLT XPath instruction cannot address a section of the document outside of the current "node" of the document as it will not be available.

While streaming processes extremely large documents, you must follow the requirements. You'll need to create streaming rules and compile options policies and check to ensure your XSLT conforms to the streaming limitations. For more information about streaming, see the Optimizing through streaming topic in the DataPower Information Center.

Multistep issues and unnecessary context

Care must be taken when defining processing policy rules to avoid unnecessary memory usage. Most actions create output "context" and it is important to realize that each new context represents an additional allocation in memory. Figure 6 shows an example of two transform actions that create context (ContextA, ContextB), which is then sent to the output stream through a results action.
In many occasions, you can use the special "PIPE" context to avoid this intermediate context creation. The PIPE context does not require separate memory for each processing step and has other performance advantages as well. While some actions require an intermediate context, and in some cases you'll need to have noncontiguous processing patterns, you should ensure that each processing policy is reviewed for optimum context usage.

Another important tool regarding context is the use of the special "NULL" context. This "bit bucket" is useful when an action does not produce meaningful output. Perhaps all you need to do is log some data, or set a dynamic route. If you are not modifying the message, subsequent actions can access the original input data and you do not need to pass it along with XSLT "Copy" statements and the unnecessary production of context.

Latency and timeouts
Latency and timeouts are important factors in memory consumption. Consider a typical scenario in which requests are being processed through DataPower and onto a backend service. Transaction rates are high, throughput is as expected. Now consider that the backend service becomes slower to respond, but it is responding and not timing out. Requests come into DataPower at the previous rates, unaware of the slowdown occurring on downstream services. But, the transactions are not completing until the response is received from the backend and potentially processed within a response rule. DataPower must maintain the request data and variables produced during response rule processing.

In addition, latencies may not be associated with the service's "backend", but by other endpoints accessed during request or response rule processing. There are a variety of interactions that may take place. Logging, authentication, orchestrations, or other integration services may be called. If they are slow to respond, the transactions are slow to complete. If transactions are accepted at a continuous rate, they begin to queue up in an active and incomplete status. These transactions hold onto resources until they complete.

Backend timeout values are set at the service. The default values are typically 180 seconds and controls initial connections and the maintenance of connections between transactions. User agent settings (which are identified from the service's XML Manager Object) are used to specify
the timeout values of "off-box" or inter-rule requests. The default value is 300 seconds. This is probably too much and more restrictive values should be used, allowing connections to fail when connections cannot be made in a realistic time. Timeouts vary between endpoint types (HTTPS, ODBC, and so on) and may be dynamically altered using extension functions. Consult the product documentation for your specific service configuration.

Timeouts may be identified by log messages and analyzed through the use of log targets, which consume these events. Latencies are potentially more insidious. You may not be aware of increases in latencies (unless you are monitoring these values). However, you may use monitoring techniques, such as SNMP monitors to query service rates and duration monitors. Some customers will utilize latency calculators through XSLT and potentially create log message, which again can be consumed by log targets for dynamic configuration or analysis.

**Throttling**

DataPower constantly monitors system resources, including memory and CPU. To avoid an excessive use of resources, throttle settings allow for a temporary hold on incoming transactions until the constraint is relieved. Using these throttle settings allow for inflight transaction to complete before additional transactions are accepted. In a typical high availability environment, transactions are processed by other appliances in the HA peer group, relieving load on the saturated appliance. Figure 8 shows the default values for throttling. In this example, when memory is at 20% of available memory, the firmware waits "timeout" seconds and then reevaluates the memory. If the memory constraint has not cleared, the firmware restarts. If at any time memory falls below the "Terminate At" value, an immediate restart occurs.
Figure 8. DataPower memory throttle settings

You can use the backlog queue to hold incoming transactions while waiting for the resource freeing. The "Backlog Size" number or transactions (currently limited to 500) can be queued for a maximum of "Backlog Timeout" seconds. If the backlog size is at its default of "0", transactions are immediately rejected during throttling. As the throttle process evaluates memory and other resources, it can be configured to produce detailed log messages. Setting the "Status Log" to "on" from its default of "off" produces messages like those shown in Listing 1. Memory, Port, Free space, and File System snapshots are captured. As with all log messages, you can send these events to a process for further processing. That process can, for example, be another DataPower service that executes XML management commands to modify the configuration settings.
Listing 1. Example of throttle status log entries

1,20130407T125910Z,default,usage,info,throttle,Throttler,0,,0x0,, "Memory(3923046/4194304kB 93.5% free) Pool(250) Ports(872/874) Temporary-FS(158771/202433MB 78.4% free) File(OK)"
1,20130407T125910Z,default,usage,info,throttle,Throttler,0,,0x0,, "XML-Names Prefix(2/65535 100.0% free) URI(83/65535 99.9% free) Local(1374/65535 97.9% free)"
1,20130407T125931Z,default,usage,info,throttle,Throttler,0,,0x0,, "Memory(3920133/4194304kB 93.5% free) Pool(417) Ports(872/874) Temporary-FS(158771/202433MB 78.4% free) File(OK)"

You now understand some of the resources available to you. You see how the memory data is used to dive into domains, services, and individual actions to see where memory resources are being consumed. Next, we’re going to discuss some tools that are important in performing dynamic analysis of transaction flows and that can be used to affect transaction rates. Later, we'll discuss critical issues of transaction flow regarding interaction with services off the appliance, such as the logging service and backend resources. You'll see how they can greatly affect transaction flow and how you can use the service level monitoring tools to mitigate these issues.

Managing services

DataPower services provide an extremely efficient processing environment through the purpose-built hardware and firmware. However, there are good reasons to control the rate at which transactions are accepted and processed. We've discussed the interrelationship between DataPower and external services. You do not want to accept transactions at a rate that exceeds the external service's abilities to process them. You may also want to offer different classes of services. Your "gold" customers may warrant a higher rate of processing than your "bronze" customers.

In this section, we'll describe some of the fundamental configuration options that you can use to accomplish these objectives. We'll describe count and duration monitors and service level management policies, which extend the monitor capabilities with more options and the ability to define multiple "rules" for complex service level monitoring.

DataPower service management options go far beyond these basic capabilities including integration with WSRR. WSRR provides organizational governance capabilities including policy definition and automatic service configuration. You can define the policies in WSRR and have the DataPower configurations automatically created. You are encouraged to investigate these capabilities and to refer to the Related topics section of the article.

Count and duration monitors

Count monitors and duration monitors provide a simple method of transaction control. Both monitors work by selective execution of a "filter action". The filter action can shape or reject transactions, or simply produce a logging message. Of course, logging messages can trigger logging events, including monitoring actions and alerts.

Shaping involves the queuing of transactions for subsequent processing. The option to shape messages should be done carefully. Shaping is used to minimize the number of transactions that must be rejected in the time of a brief and temporary network spike. For example, if a...
When a backend server is experiencing a period of high utilization and begins to show increased latency, DataPower can limit the traffic to this server and the server may recover. If the period is relatively brief, it may be preferable for DataPower to queue a few transactions in memory rather than rejecting them. The transactions can be released when the backend latency has decreased. This can have the benefit of reducing the number of errors seen by the clients.

The drawback to shaping is increased memory utilized to hold (queue) the transactions. If the spike is too long, resources may be constrained before transactions can be released. Once accepted into the shaping queue, you cannot cancel a transaction. The queue size is fixed and you cannot configure it. These are important considerations to take into account when choosing to shape traffic.

One important factor in the duration monitor filter calculation is that duration monitors measure the average time for transactions to complete. It is important to note that the algorithm only considers the average time of the last several transactions. It is not an absolute limit of a single transaction. A common use case is to configure a monitor that generates a logging event if the average total latency of the service is climbing above some threshold.

Figure 9 shows the definition of the filter action. Within this configuration, the filter action is defined, which is "Shape" in this case.

**Figure 9. Monitor filter action rejecting messages**

The messages to which filter rules are applied can be selective. You can use a variety of conditions to determine the characterization of messages used for filter calculations. For example, the URL of the input message, HTTP headers, and HTTP methods might be part of the conditional filtering. Figure 10 shows an example of selecting only those messages whose HTTP method is POST.
Figure 10. Message type definition selecting POST HTTP method

Combining the message type and filter action produces the count or duration monitoring object. In the example in Figure 11, POST type messages are counted. When they exceed 100 TPS (1000 Millisecond interval), the messages are "shaped" or placed into the temporary queue and executed at a controlled rate. The threshold calculation includes a "Burst Limit" value. This value allows for an uneven transaction rate and accommodates a calculation in which "unused" counts in previous intervals are allowed in successive intervals. The general best practice is to use an interval of at least 1000 milliseconds and a burst rate of 2 times the rate limit.

Figure 11. Count monitor with message type and filter action (see enlarged Figure 11)

Service level monitoring

Service level monitoring (SLM) extends count and duration monitors by providing a more selective basis for transaction analysis and the ability to combine SLM rules or statements to develop complex monitoring processes.
SLMs are configurable as a processing action within the processing policies of Multi-Protocol Gateways and Web Service proxies. An SLM action specifies an SLM policy object, and each of these objects is composed of an ordered sequence of one or more SLM statements. Each SLM statement defines a separate set of acceptance and enforcement criteria as well as the action to be taken. SLM policies provide the option to execute all statements, to execute statements until an action is taken, or to execute statements until the policy rejects a message. Figure 12 shows a configured SLM policy containing a single SLM statement.

**Figure 12. SLM policy with a single policy statement (see enlarged Figure 12)**

Each SLM statement provides a number of options to be configured:

- **Credential and resource classes**: These specify criteria used to select to which incoming messages the statement will be applied. This may include user identity details from an authentication action, transactional meta data, such as URL or transport headers, or custom information provided through a user-written XSLT.
- **Schedule**: This specifies the time frame when the statements will be applied. This provides for the ability to preconfigure events such as downstream application maintenance or "Black Friday" shopping events.
- **SLM action**: This specifies the action to take if an incoming message exceeds the statement's threshold, which is typically either notify, shape, or reject. These are similar to the count and duration monitor actions.
- **Threshold interval length**: This specifies the length of the measurement interval in seconds.
- **Threshold interval type**: This specifies how intervals are measured. Intervals can be fixed, moving, or concurrency based.
- **Threshold algorithm**: This specifies how incoming messages are counted within a threshold interval. Typically, a simple "great-than" algorithm is used to cap transaction rates. However, a more complex algorithm such as "token-bucket", which is similar to the Monitor "Burst Rate" calculation, is also available.
- **Threshold type**: This specifies how incoming messages are applied to a threshold interval, either by counting or by tracking latencies.
- **Threshold level**: This specifies the trigger point where the action is executed.
Figure 13 shows the SLM statement form. You can combine multiple SLM statements to provide for sophisticated service level agreements and throttling procedures. By configuring a set of SLM statements, each of which is tailored to handle a particular situation that can lead to memory or resource exhaustion, the appliance as a whole can be better protected from the negative impacts of anomalous situations within clients, side services, and back-end servers. The SLM design strategy is to limit the flow of incoming messages when there is the potential for slow transaction processing times leading to a build-up of transactions being processed, since each in-process message consumes resources of the appliance.

The following sections describe how you can configure each of the SLM options to handle these types of situations.

**Combining monitors and SLM**

There are occasions when combining SLM and count or duration monitors provide the most effective transaction control. Count monitors do not consider the latency of a transaction. Count monitors always use a rate algorithm, which only counts incoming requests. For example, if the monitor is enforcing a limit of 10 TPS, but the backend is slow and is taking 120 seconds to
complete, the appliance can have thousands of simultaneous transactions active. SLM, when using algorithms such as greater-than or concurrent, considers the latency so SLM can protect against slow backends.

**Note**

A best practice is to use monitors as a gross admissions control algorithm while allowing SLM to handle the finer-grained details of managing resources.

SLM is, in essence, a transform and uses resources in its processing. An attacker can flood a box with simultaneous connections and the resources become constrained before SLM calculations have even started. Count monitors are very lightweight so they can handle a connection flood.

**Implementing a service level management**

Now that we've described some of the basic components of transaction management, let's discuss some simple ways to better regulate transaction flows. We've described monitors and SLM policies and how you can easily use them to monitor transactions following through to backend services, and how you can use these to control latencies in backend services. You can also use monitors and SLM policies when interacting with "off box" services. We mentioned in our introduction how DataPower can interact with many different end points and we've described a logging service as a good example. What happens when that logging service become slow to respond? Transactions begin to queue up in DataPower and consume resources. So, let's use these techniques to control that.

Rather than accessing the logging service directly through a results action, we'll create a "façade service" and, within it, we will apply monitoring capabilities. This allows for the ability to monitor, shape, or reject requests that are becoming too slow to respond. The façade service is necessary to encapsulate the monitors as the results action by itself does not provide this capability. If you are using firmware version 5.0 or greater, you can also use a "called rule" in place of the façade service. The called rule contains the monitors in this case. Figure 14 shows an example of our architecture.

**Figure 14. Facade service as an “off box” gateway**

![Diagram of facade service as an "off box" gateway](image)

**Creating the façade service**

Create the façade service on the same device to minimize network delays. It can, however, be in another domain on the device. The SLM resource class should be concurrent connections.
This is an important point. Other algorithms have a potential vulnerability to a slow backend. But, concurrent connections do not because it is an instantaneous counter.

The façade service’s front side handler (FSH) is a simple HTTP connection. In this case, do not use persistent connections. There are several reasons for this:

- First, over the loopback interface, there is no resource or performance penalty for not using persistence.
- Second, when using persistence the appliance caches some memory after each transaction, which can increase the overhead of the service. Therefore, as there is no benefit, do not use persistence.

Figure 15 shows the simple façade service (or possible called rule). Again, all we are doing is encapsulating an SLM policy within the path to the logging service.

**Figure 15. Façade service rule with an SLM policy**

The SLM policy is demonstrated in Figure 16, which shows the policy with one rule and the details of the resource class (using concurrent connections), and the throttle action, which rejects messages. The policy rule is using a fixed interval of one second with a "count all" threshold of 20. That is, allowing concurrent transactions and rejecting those in excess of 20.

**Figure 16. SLM policy to reject concurrent transactions greater than 20**
Figure 17 illustrates the configuration change to the main processing policy. In the original rule, transactions went directly to the logging service; in the second, they are sent to the façade service on the loopback (127.0.0.1) interface.

**Figure 17. Policy rule before and after using the façade service (see enlarged Figure 17)**

It's as simple as that. We have now altered our configuration to monitor transactions to the logging service. When they are excessively slow, we reject the entire transaction.

In summary, some of the best practices for using the façade service are:

- The backend can be any protocol; it is HTTP for this example.
- All actions in the rules must be synchronous.
- The response message type should be pass-through (unprocessed).
- The request type should probably be non-XML (preprocessed) for minimum overhead.
- All other settings are set as defaults.
- If necessary, you may want to explore streaming and flow control. This is useful if you are using an asynchronous action to send large amounts of data.
- The request rule should have a single action, which is SLM.
- The input and output of the action should be NULL.
- The SLM policy should have a single statement that uses the concurrent connections resource.
- The statement should reject the extra transactions.

**Demonstration of resource utilization with the façade service**

Having created the façade service, let's examine the effect it has on service resource utilization. We'll process transactions through a service, which directly access the logging service and one that uses the façade as a gateway, and which uses the concurrent connection SLM policy. We'll use memory status as an indicator of system utilization.

In the first test, we'll use **Apache Bench**, a free and simple tool to process a series of transactions at various rates of concurrency. We have created a logging service with a built-in latency to demonstrate slow-to-respond services. As you can see in Figure 18, as the transactions begin to slow down, they are queuing up and consuming memory.
In the second test, we'll use Apache Bench again to process a series of transactions. However, in this example, while the logging service is again slow to respond, the SLM policy is rejecting transactions and the effect within DataPower is a dramatic change in resource consumption. Figure 19 shows that the memory never goes below 95% free. This rejection of transactions is typically advertised through a logging message, alerts, or other monitoring tools. Administrative staff, being alerted to the delays in the "off box" service may respond to the latency issue. If the issues are systemic, for example, "Black Friday" sale spikes, additional DataPower resources may also be allocated to handle these periodic traffic spikes.
Conclusion

In this article, we described the central position that DataPower often takes in policy enforcement and service integration. We've described how this architecture is affected by latencies within interactions with "off box" services. Service latencies can, if unregulated, have a deleterious effect on DataPower transaction processing. DataPower provides several methods for resource monitoring, and we demonstrated the ability to analyze resource utilization at the system, domain, and service, and down to the specific actions within a processing policy.

We described some of the fundamental transaction management options, including count and duration monitors and service level management, and how you can use them to regulate and smooth transaction flow and to mitigate latencies exposed by services with which DataPower interacts. We also mentioned the more advanced governance capabilities available through WSRR.

Finally, we demonstrated how you can use these techniques in a specific use case, in which an integration service (logging service) becomes slow to respond, and how using the façade service technique provides a more resilient and effective DataPower implementation.

Acknowledgements

Many of our IBM colleagues assisted in the preparation of this article. The authors would like to thank Barry Mosakowski, David Shute, Carol Miller, and Daniel Badt.
Related topics

• **developerWorks articles and DataPower publications:**
  - IBM WebSphere DataPower SOA Appliances product documentation
  - IBM Redbook: WebSphere DataPower SOA Appliance Handbook
  - IBM Redbook: Strategic overview of WebSphere Appliances
  - Monitoring WebSphere DataPower SOA Appliances
  - IBM Redbook: DataPower SOA Appliance Administration, Deployment, and Best Practices
  - WebSphere DataPower SOA Appliance performance tuning
  - End-of-service dates for the Hardware Generation Machine Types (M/T): 7993, 9235, 4195, 7199, and 7198
  - IBM WebSphere DataPower SOA Appliance Firmware Support Lifecycle
  - WebSphere DataPower Information Center: Optimizing through Streaming
  - Enforcing Service Level Agreements using WebSphere DataPower, Part 1: Applying the SLA Control File pattern
  - SOA governance using WebSphere DataPower and WebSphere Service Registry and Repository, Part 1: Leveraging WS-MediationPolicy capabilities
  - developerWorks WebSphere DataPower zone

• **Tech notes:**
  - Why does memory % differ for "show memory" and "show load" when using DataPower?
  - Total amount of memory of DataPower 7198/9 devices
  - IBM WebSphere DataPower Appliances firmware V5.0 adds support for extended memory, OAuth 2.0, and enhanced governance and SLA management features
  - IBM WebSphere DataPower Service Gateway XG45 and WebSphere DataPower Integration Appliance XI52 virtual editions provide flexible deployment options
  - WebSphere DataPower spikes with high CPU when WebSphere MQ connection is unavailable
  - Gathering WebSphere DataPower CPU and memory information
  - How does the front side timeout work?
  - What IBM DataPower timeouts are used in specific configuration methods