IGC White Paper:

Brava!® scalability and fault tolerance

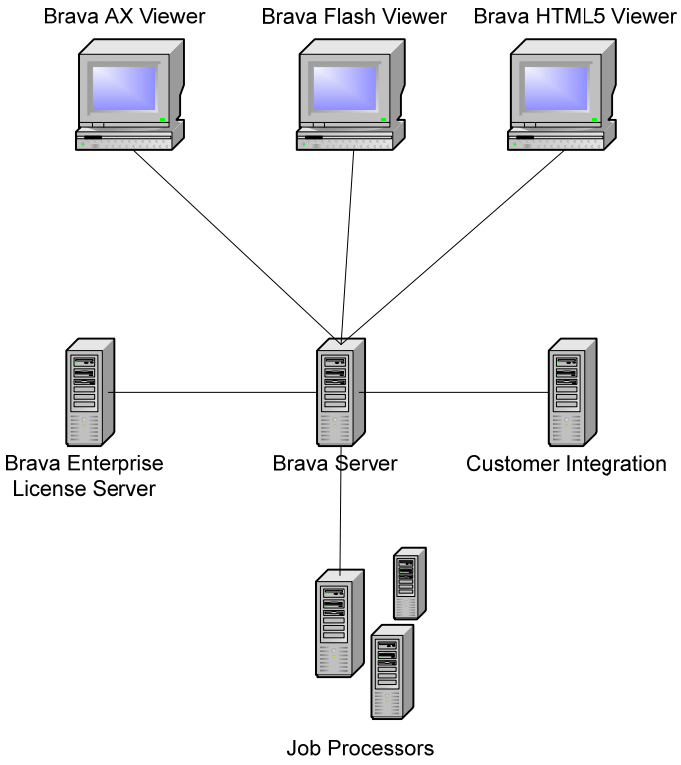
1 Brava!® scalability

1.1 Brava scalability and fault tolerance introduction

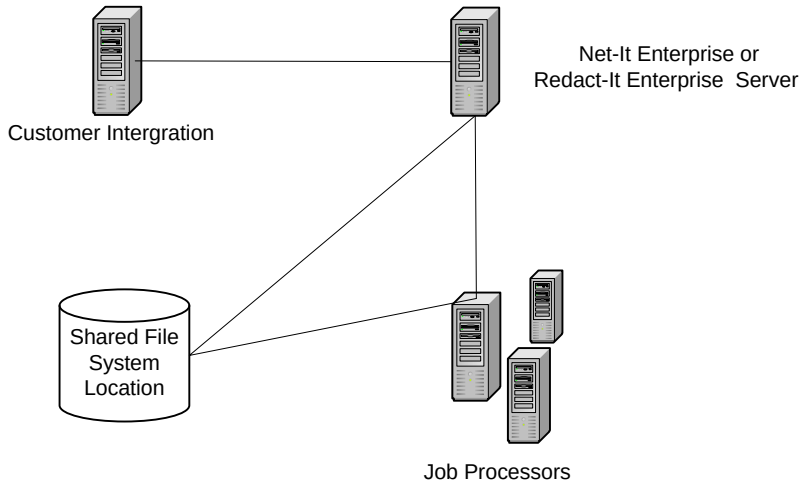
Brava scalability touches many parts of the system, including the integration in which Brava runs. This section covers scalability and fault tolerance of the Brava system itself. Integration scalability is covered in a separate section.

1.2 Component overview

Here are two simplified diagrams of the components that make up a Brava or Net-It® Enterprise/Redact-It® Enterprise installation.



Brava Enterprise components



Net-It Enterprise/Redact-It Enterprise Components

1.2.1 Brava Enterprise server

There are two different types of servers for Brava: a Servlet-based server, called the Brava Server or the Servlet Server, and a .NET-based server, called Brava Enterprise.NET (BEN). At the time of this writing, there are slightly different feature sets in each product, namely that the 7.1 Servlet Server has shipped with support for an Oracle-based database cache. Brava.net is currently at version 7.0.1, and will not support a database cache when 7.1 ships.

The Brava Enterprise Server is responsible for many things:

- It exposes an HTML interface used by the ActiveX control to display files and manage markups
- It also exposes a RESTful interface used by the Flash viewer and upcoming HTML5 viewer
- It caches renditions of published files and manages the publication of files
- It can be used to store and manage markups
- It is responsible for organizing licensing by talking to the Brava Enterprise License Server (BELS)

The Servlet server runs on Windows and Linux. Brava.net runs only on Windows servers.

1.2.2 Brava Enterprise License Server

The Brava Enterprise License Server (BELS) is responsible for licensing the Brava Server and the clients. It runs on a Windows server, and uses .NET. Only the Brava Server communicates with BELS. The various clients and the Job Processor (JP) do not communicate with BELS. Communication is over a dedicated, configurable port. Multiple Brava Enterprise License Servers can be configured for purposes of fault tolerance. The Brava Server polls a list of configured Brava Enterprise License Servers to accomplish this.

1.2.3 Brava Job Processor

The Brava Job Processor (JP) is the component of the system responsible for rendering native content into representations appropriate for the various IGC clients. It processes multiple files at one time, and multiple JPs—each on separate systems—can be installed to provide additional capacity and fault tolerance. The JP is a .NET and Win32-based system that runs on a Windows server.

1.3 File types

The Brava System (the Brava Server and the JP) processes different types of files in different ways, and this can affect scalability and fault tolerance. See section Job Processor Types of job processing.

Besides client-side publishing (see the next section), the Brava server makes decisions on how to process a file based on the extension supplied to the publish request. The first decision made by the Brava Server is which queue to place the file in. This is controlled by settings in the Server.properties. Furthermore, the Brava server may decide to request a single-page publish (of the first page in the file) for those file formats that support it. This is also controlled in the Server.properties file. Note that these decisions only control which queue the job request appears in. The JPs are then configured to process the jobs in one or more of the queues. See the Brava Enterprise documentation for an explanation of queues as well as Appendix A, Brava and Net-It queues.

1.4 Client-side publishing

In some installations, it can be useful to have certain file types published on the client, rather than on the server. This is useful, for instance, when the system is viewing files that are both small and infrequently viewed—when a bank uses Brava to review images of cashed checks, for example. In that case, the integration can request that the client publish the file rather than the server (via the convertonclient=true parameter). This is only supported by the ActiveX viewer, and there are limitations with some formats (e.g., AutoCAD drawings with external xref files will not convert on the client). Client conversion can be used selectively, based on logic in the integration. If client conversion is used, the entire native document is streamed to the client's machine and published locally using IGC drivers. This may result in slightly more bandwidth than if the file is converted on the Brava Server and cached. When loading from the Brava Server cache, the client will only request the pages needed, so a large, multipage document might result in slightly less bandwidth. This concern is minor, though, and will likely make no difference in the calculation of bandwidth, as that calculation likely assumes that the user will view the entire document.

Client-side publishing has performance and scalability impacts. Since the entire file is streamed to the client, there may be more bandwidth required than when using server-side publishing. If the files contain only one or two pages, the difference here is negligible. Client-side publishing also prevents the server from caching the published document, so if the same document is viewed multiple times, server-side publishing is recommended, since the cost of document conversion is incurred only once, upon first document view. The actual job of conversion or publishing of the document on the server or client requires similar work and time on similar hardware.

1.5 Overall architecture concerns

Generally, there are two major scalability concerns in Brava. The first is the Brava Server's ability to handle the expected client and integration traffic, and the second is the ability of the JP to publish files in a timely fashion. The first issue is solved by using one or more servers (and servlet containers like Apache Tomcat, WebSphere, WebLogic, etc.) suitable to the expected load. The second issue is solved by installing the JP on a sufficiently powerful system and/or adding additional JPs.

Bandwidth between the Brava Servers, the cache technology and location (e.g., file system versus Oracle database) and the JPs is an important concern as well.

1.6 General Brava Enterprise Server

1.6.1 Brava Enterprise Server Bandwidth requirements

Multiple bandwidth usages must be considered when planning your Brava Server.

- **The bandwidth required retrieving the source file.** The source file is either provided during the call to publish the document, or the Brava Server will retrieve the file from a specified URL. In both cases, the source file will be inserted into the cache during conversion. Either method requires sufficient bandwidth to bring the file to the Brava Server in a timely fashion.
- **The Job Processor will read the source file and write the results from and to the Brava display list cache.** Sufficient bandwidth must exist for all the JPs to allow the JPs to read and write concurrently.
- **The Brava Server serves up published files to the various clients.** This bandwidth is from the Brava Server web host (IIS, Apache, etc) to the clients, potentially running outside the corporate network.

1.6.2 Display list cache scalability

Brava Enterprise uses a display list (DL) cache to store the result of “published on the server” jobs. Files are published on the server when 1) the ActiveX Client, with the convertonclient=false parameter, needs to view a file and 2) when the Flash and HTML5 clients need to view a file. There are two implementations of the DL cache: shared and unshared. Shared in this sense is relative to other Brava Servers. Both cache implementations are shared among clients that hit the same Brava Server. The shared cache is implemented with the Oracle 11gr2 database, while the un-shared cache is implemented with the file system. The scalability of the DL Cache is thus directly related to the Oracle Database Management System hardware/software/network configuration for the shared cache and to the network file storage hardware/software configuration for the un-shared cache. The cleanup of the shared cache is up to the database administrator, as Brava Server has no automatic cleanup in this implementation. Simple stored procedures can be executed by the administrator to accomplish this. The cleanup of the un-shared cache is executed implicitly and is controlled by many parameters found in a Brava Server configuration file.

1.6.3 Markup storage

The core implementation of Brava Server stores and retrieves markup files. In the shared-cache implementation, markup file data is stored/retrieved to/from the Oracle Database. In the un-shared implementation, the markup file data is stored/retrieved to/from the file system.

1.7 Brava Enterprise Servlet Server

The host server, along with its servlet container, should have sufficient bandwidth to handle the expected calls to publish and view each file. In both the ActiveX and Flash client cases, the server will likely be distributing large binary files from the Display List Cache to the clients. The main servlet containers Brava Server supports are Tomcat, WebSphere and WebLogic.

1.7.1 Server Load Balancing

Brava Server works well with load balancers. In a shared-cache mode, no special integration code is necessary to dispatch the servlet invocations to the various Brava Servers. This is due to the "client/server session IDs" being shared across all Brava Servers in the system. However, in the un-shared mode, the session IDs are tied to specific Brava Servers, which requires the integration to route the subsequent servlet invocations to the same Brava Server that passed out the session id to the client.

1.7.2 Web server

In our performance testing, we have seen dramatic improvements by utilizing the Apache HTTP Server instead of the Web Server found in Apache Tomcat Servlet Container. Furthermore, isolating Apache on a separate machine also improved performance significantly.

1.8 Brava Enterprise .NET

1.8.1 IIS Considerations

The IIS server chosen to host Brava Enterprise.NET (BEN) should have sufficient bandwidth to handle the expected calls to publish and view each file. In both the ActiveX and Flash client cases, the server will likely be distributing large binary files from the Display List Cache to the clients.

At this time, the BEN server does not support the shared cache, so while you can install more than one BEN server for fault tolerance, they will maintain separate caches of each set of published artifacts. Any load balancing needs must be handled by the user's integration.

1.9 Brava Enterprise License Server

BELS communicates with all Brava Server instances, serving up both client and server licenses. The bandwidth between BELS and the Brava Servers will therefore see requests whenever a client connects to the Brava Server.

It is possible to configure Brava Server to connect to multiple license servers to provide redundant failover capability in the case of accessibility loss to any one license server.

When a client license is required due to a user request to view a document using Brava, the Brava Server will attempt to obtain a license from one of the listed license servers. If communication with the license server fails, another will be tried until either a license is obtained or the list of license servers has been exhausted. If communication with a license server is lost, the Brava Server should regain access shortly after communication has been restored without requiring a restart of the Brava Server.

1.10 Network communication considerations

1.10.1 Communications between the integration and the Brava Server

The bulk of communications between the integration and the Brava Server occurs when supplying the native file to the Brava Server for conversion. This transfer happens either when the integration pushes the file to the Brava server or when the Brava server pulls the file from the Integration. In either case, the entire file is eventually transferred over the wire to the Brava Server, and the native file is stored in the display list cache until conversions are complete, at which time it is deleted.

Therefore, there must be sufficient bandwidth between the integration and the Brava Server to deliver the files to the Brava Server in a timely fashion under the heaviest expected load. This should be measured by simultaneous first requests to publish files, as once published, the file is cached and the Brava Server no longer needs to retrieve it.

1.10.2 Communications between the Brava Server and the JP

When publishing a file, the JP reads the file from its location within the display list cache. This generally requires that the JP request the contents upon publication of the entire file Furthermore, the published artifacts will be placed in the display list cache when publication is complete, thus requiring another "file size" amount of data to be transferred from the JP to the cache. Generally speaking, the XDL representation of a file is equivalent in size to the native file. A good rule of thumb: Each conversion on the JP will transfer twice the file size number of bytes across the wire.

Note that if you are using single-page publishing, the file is read a second time (although perhaps not all of it) at least once by the JP, and the results are written back into the cache. These results are much smaller than the original file, as they represent only a single page of the file.

1.10.3 Communications between the Brava Server and the client

IGC very rarely sees network bandwidth as a constraint in even highly loaded Brava installations. This section provides some guidelines on what is sent across the wire and how big the load is.

1.10.3.1 ActiveX Client

The ActiveX client can operate in two modes: client conversion and server conversion.

In the case of client-side conversion, every client that attempts to view a file will have that entire file sent across the wire to the browser for conversion. Thus, the bandwidth required is the number of simultaneous viewing clients times the size of the file. Furthermore, each client must download the drivers to perform the conversion, but this download only happens once per client.

If you are using server-side conversion, then the client will request the published XDL artifacts from the display list cache (via the Brava Server), and the transferred amount will be, at most, the size of the published artifacts. This published content is delivered in an on-demand manner to reduce bandwidth. For example, if only one page is viewed by the client, then only one page of published content is downloaded. If the client performs a more complex operation like a document-wide search or publish, then all pages of the published content will be downloaded. Note that if the thumbnail panel is visible, all the thumbnails exposed within it will cause downloads of corresponding page published content.

1.10.3.2 Flash client

For the Flash client, the JP produces high resolution PNG or JPG resolutions of the pages in each file, and those files are transferred, as needed, to the Flash client. These files are often substantially larger than the vector representation of the file (say, a Word file), and sufficient bandwidth must be present to move those images (and smaller thumbnails) for each simultaneously viewed page. Note that if the thumbnails are visible, or the view mode is in a page spread mode, then more than one page of published content will be downloaded.

1.11 Physical storage considerations

1.11.1 Brava Server storage considerations

The Brava server requires storage for two things: markups and the display list cache. Markup storage by the Brava Server is optional (they can be stored by the integration).

There are two kinds of display list caches in Brava Server, the file system (un-shared) cache and the shared cache (note that the shared cache is not supported in BEN at this time).

Both caches are used to store the native file during conversion, and then to store the published renditions of the file. Native files are only stored temporarily, but renditions are stored until the rules in the server's configuration indicate the file should be removed.

The size of the file system cache can be controlled by the server.properties file, so storage requirements are relatively straightforward to control.

Automated cache cleanup is not available for database cache sharing as it is with file system caching. Oracle database administrators can manually clean up cache entries based on access list date stamps. Multiple stored procedures are available to the administrator to control the size of the shared cache in the database.

1.11.2 JP storage considerations

The JP does not store native or published renditions of documents. It only requires temporary storage space for intermediate artifacts. Various process monitors can be configured to make sure that temporary files are properly destroyed and don't take up too much space. See the Brava, Net-It Enterprise or Redact-It Enterprise user's guides for details on configuration.

2 Job processor

The job processor (JP) is key to the scalable conversion of documents. When used with Brava, the JP's job is to publish file types to IGC internal formats (XDL or CDL) or to raster formats for viewing by the Flash or HTML viewer. When used with Net-It Enterprise or Redact-It Enterprise, the JP has more capabilities that are highly dependent on what the integration is trying to achieve. These scalability notes apply in both cases.

It is very important to ensure that each JP is on dedicated software when attempting to provide more throughput with one or more JPs. If the JP is installed on a system that has other responsibilities, then the JP can't be scaled up to use all available resources, and it is therefore fundamentally constrained from reaching its full performance potential on that hardware.

2.1 JP scalability

The JP scales in two ways. It can scale up with more hardware on a single machine (subject to the limitations detailed below), and it can scale out to additional systems. Furthermore, when scaling out to multiple systems, the workload of each JP can be varied so that the number of JPs and responsibilities of each can be tailored to provide a responsive environment that matches the kinds of documents specific to an individual customer. The types of technology used to process your files (IGC native drivers, automation publishing or Outside In-based drivers, see Type of job processing, below) also affect your decision to choose scaling up or scaling out, or both.

2.1.1 Scaling up

Scaling up is useful when more powerful hardware is available. Files published with IGC drivers or Outside In drivers benefit from scaling up. Individual files will publish more quickly, and multiple jobs can be published in parallel.

Each JP requests work from one or more queues and uses a single process to service each request. The number of simultaneous processes is primarily controlled by the number of cores in the system and the available memory on the system. A 16-core system can process more simultaneous requests than a four-core system. Installing the JP on machines with more cores will produce nearly linear performance improvements.

Generally speaking, you can start with between four and six threads per core. Therefore, if you have a quad core machine, start by setting the total count of all threads in jobprocessor.config to equal 16–24. Testing the system with representative samples of files should then be conducted to further tune the system. As you increase the number of threads, you can graph the number of threads versus the pages/second throughput. In most cases, this graph will peak around four times the number of cores and then flatten

out as more threads are added, eventually decreasing as network, memory or disk I/O become a bottleneck.

It is important to note that the JP is designed to work on dedicated hardware (physical or virtual), and increasing thread counts will eventually consume all the CPU and memory resources of the machine. This is by design. If you must run a JP on a machine that has other duties, then the number of threads per core must be reduced to allow for other software on the system to be adequately responsive.

Note that as of the 7.1 product family, you can increase the doc thread queue count above one. The doc queue is responsible for conversion of files via IGC's automation publishing. This queue does not scale quite as well as native driver conversion because the system must enforce serialized access to the various COM automation interfaces it uses, thus introducing an unfortunate bottleneck. Nonetheless, performance improvements can be seen by increasing this queue's thread count, as the pre- and post-conversion tasks are parallelized.

2.1.2 Scaling out

As throughput demands increase, a single JP may not be sufficient to handle the load. In this case, you must add JPs to the system. Since the communication of jobs sends very small amounts of data (the JP requests a job that is merely a few lines of text), JPs can be added in great number without adversely affecting the queue servers. Besides increasing overall throughput, scaling out also improves performance for file types processed with Automation Processing (see below).

Each JP can be configured as described above, in scaling up. However, additional end user performance improvements can be gained by tailoring JPs to do different things. This is especially true in the case of Brava Enterprise and its single-page publishing feature. See single-page publishing below.

2.2 JP fault tolerance

JP fault tolerance is achieved in two ways, just like scalability. On a single JP, each conversion job is handled by a separate process, so if that process fails or times out, it doesn't affect other conversion activities.

Adding JPs to the system increases this fault tolerance and provides robust fault tolerance in the case of network or hardware failure.

2.3 Types of job processing

2.3.1 Native IGC file types

Many file types that are opened by IGC's technology use "native" drivers. Native drivers, like dwg2dl.dll and pdf2dl, are drivers—maintained by IGC (potentially using licensed third-party technology)—that know how to load one or more file types. Native drivers have the most scalability options because all of them scale with multiple conversion processes on a single JP and have no inter-process communication concerns. All recommendations for scalability apply to file types opened with IGC native drivers.

2.3.2 Automation-based publishing

The JP can open any file type with a Windows shell-related "PrintTo" command. Furthermore, using IGC's BI2DL, the JP can publish Office documents (Word, Excel, Powerpoint) via Office's export to XPS capability. While this method provides very good fidelity, it does have a serialization requirement that constrains scalability on a single machine.

Because of the nature of Microsoft's Automation API used to drive Microsoft Office, all calls to the office API (to open or export a document, for instance) must be serialized. The JP and associated technologies (specifically, Net-It Now) ensures this serialization. As of the 7.1 family of products, the JP's document queue count can be increased beyond one, but the jobs will still be serialized during the production of the XPS or print publishing actions. All other activities taken by the JP (loading the XPS or EMF representation of the file, exporting to PDF, redacting text, etc.) will happen in parallel.

Therefore, if your JP is using "print publishing" technology (BI2DL) and not INSO, while some benefit is seen when increasing the number of simultaneous threads for office file types, performance increases will plateau sooner than memory, CPUs or bandwidth would suggest. As such, additional JPs may be required to achieve your performance goals.

2.3.3 Oracle's Outside In-based publishing

The JP can use Oracle's Outside In technology to process certain file types. Outside In publishing is often referred to as Inso in this and other IGC documents for historical reasons.

When you configure a JP to use Inso (via the myrdrv.ini file), all of the scalability solutions presented above are valid, as the JP can use Inso fully concurrently with no serialization or other limitations.

You must choose to use either Inso or Automation (print) publishing, as the two technologies result in published content with different coordinate spaces. Because of the different coordinate spaces, markup files cannot be shared between renditions of the same file from the two different conversion technologies.

2.3.4 Single-page publishing

In the case where the JP is servicing requests for the Brava server, it's often useful to configure a single JP to do nothing but single-page publishing. This results in a much more responsive client experience in heavily loaded environments.

When viewing documents with the Brava Enterprise ActiveX viewer, viewing most file types will create two publish requests: one to view just the first page of the file (and destined for the *single* queue) and one for the entire file. Furthermore, until the entire file's conversion is complete, the ActiveX control will request individual pages as the user pages through the file. Therefore, scalability and performance improvement can be achieved by creating machines that run JPs capable of processing only single-page publish requests. To do this, simply remove all thread types from the Jobprocessor.config file except for thread.single, and set thread.single to be four times the number of cores on the machine. On the other JPs in the system, remove the thread.single from each jobprocessor.config file. Controlling

which file types result in single page publishing requests is done in the server.properties files. See the Brava documentation for details.

2.4 Multiple job processors

Multiple job processors provide several benefits. They can provide fault tolerance in the case of machine or network failure. They can also provide identical capabilities and therefore increase throughput. Lastly, multiple JPs can be configured to provide different capabilities (like the single-page publishing dedication mentioned above). These ideas can be combined, with multiple single-page publishing JPs and multiple general JPs.

If your system or integration has metadata or other knowledge about the kinds of files to be published, you can configure queues and JPs to match the expected requirements by file types. Certain JPs can be configured just to process a single type of file (say, Office docs, or perhaps a certain kind of CAD file, or even TIFF processing from a scan engine).

3 Integration scalability—info for integration designers

The design of the integration to Brava influences the scalability and responsiveness customers experience when they use Brava. This section talks about the options to change how Brava publishes and views files and the effect of those decisions on the system's scalability.

3.1 Client-side publishing

As mentioned above, Brava's ActiveX client can "client-side publish" many files types. When this happens, the native file is sent to the client machine and published on that machine. This is suitable for smaller files and for cases where caching the viewed content is not necessary. If an integration uses client-side publishing, the Brava server does not invoke the JP to publish the file, and nothing is stored in the cache, so the Brava server component's load (in memory and CPU) is lower. The client can be configured to either retrieve the native document directly (clientretrieve=true), or ask the server to retrieve the file (clientretrieve=false). Each of these decisions has consequences not only on security and access but on server bandwidth requirements as well.

If you use client-side retrieval of the document, the document is requested via URL and the bandwidth required to serve up that document is related to the document's storage, likely your integrations vault of some sort. If the server is instructed to retrieve the document, then the Brava server will request the document from the URL and then stream it to the client. Therefore, bandwidth is a consideration between both Brava Server and the integration, as well as between the client and the Brava Server.

3.2 Providing documents to the Brava Server

When an integration calls the Brava Server's publish API, it can provide the actual native file in one of two ways. It can provide them in line with the call via HTTP PUT, or it can supply the URL to the file, in which case the Brava server will attempt to fetch the file from the URL and place it

in the cache. Both methods result in about the same bandwidth use. If you supply a URL for the file, the Brava server must have permission to fetch the file.

3.3 Pre-publishing documents

Many integrations will "pre-publish" files in anticipation of their later view by the Brava Server. This is useful for the ActiveX client and can make the system more responsive. It's up to the integration to decide if a file should be pre-published, and if so, when.

Pre-publishing is simply when an integration calls the Brava Server's publish method in response to some event that will likely require the file to be viewed. This can be simply uploading the file into a vault or perhaps the viewing of some associated file.

To pre-publish a file, simply call publish on the Brava Server as you would if you were about to launch the viewer. Remember that the docid and version subsequently submitted to view the file must match those used to publish the file.

4 Net-It Enterprise.NET/Redact-It Enterprise Scalability

Net-It Enterprise.NET (NIE) and Redact-It Enterprise (RIE) have the same scalability and fault tolerance schemes available.

4.1 Queue server

The NIE/RIE queue server is an ASP.NET and Windows service (written in .NET) that runs on Windows servers. They have two main features. The first feature is the ability to queue jobs into various queues (as defined in server.properties), and the second feature is the optional enhanced directory monitoring.

Jobs in the queue are simple sets of name/value pairs, and as such do not require much space. They are stored on disk during processing, and are typically only a couple of KB in size, so even moderate hardware can handle millions of jobs. The actual source and output files are not kept in the queue.

4.2 Job processor

Aside from the comments on single-page publishing, the notes on JP scalability under the Brava section above apply to the JP when running as part of NIE or RIE. See the section on network communication considerations above.

4.2.1 General scalability

In the case of the JP running for NIE or RIE, further thoughts must be given to the work being done by the JP for each job, as the JP in NIE/RIE offers more features, like redaction, thumbnail generation, creation of markup files and exporting to various output formats. In general, these additional job options scale in the same way Brava does, though they will produce more network traffic if you request many output options for a given job. See the section on network communication considerations above.

4.2.2 Job Getter scalability

The JP uses Job Getters to retrieve jobs from the queue server or from a file system location (see the NIE/RIE SDK). Third-party integrators can also create their own job getters for specialized use. The JP allows you to use multiple job getters at the same time, but note that this increases the number of simultaneous jobs linearly with number of job getters, so you must consider the number of cores on your JP and adjust thread counts accordingly to deal with multiple job getters.

5 NIE/RIE integration scalability

5.1 Integration design and development

Your integration will generally construct a job request (a set of name/value pairs) and submit that job to the queue server via push.aspx. The integration should provide a URL that will be called asynchronously when the job completes. That URL must be able to process multiple requests as jobs may finish simultaneously and out of order from how they were submitted. Therefore, the system hosting that URL must scale with your JPs.

5.2 Providing documents to the job processor

The source files being supplied in a job are not passed to the JP. Their locations are included in the job file, and the JP will read those files when it loads the files. This includes source files, markup files, raster images for stamping and redaction scripts. Since the JP processes files simultaneously, the JP will read source files simultaneously. This mainly requires that the bandwidth between the JP and the file storage location is sufficient to provide data in a timely fashion.

6 Measuring JP scalability

To get the most out of a JP, it's important to measure its performance on representative hardware and with a sample set of documents that are similar to what you expect to see in production.

This section shows how to do this, using the Net-It Enterprise test client. The test client is available from IGC support.

6.1 Select sample files

In this example, we will take a collection of PDF files and publish them to XDL. This is essentially simulating the way Brava uses a JP. To test Redact-It Enterprise, you might select Office documents and publish to PDF while including a redaction script.

The document set is 113 documents, 2,951 total pages, the biggest single document is 540 pages, and there are a range of page in the 10s, 50s and hundreds of pages.

The important thing here is to keep the sample set the same through the entire exercise, and that the samples are similar to production files.

6.2 Determine job parameters

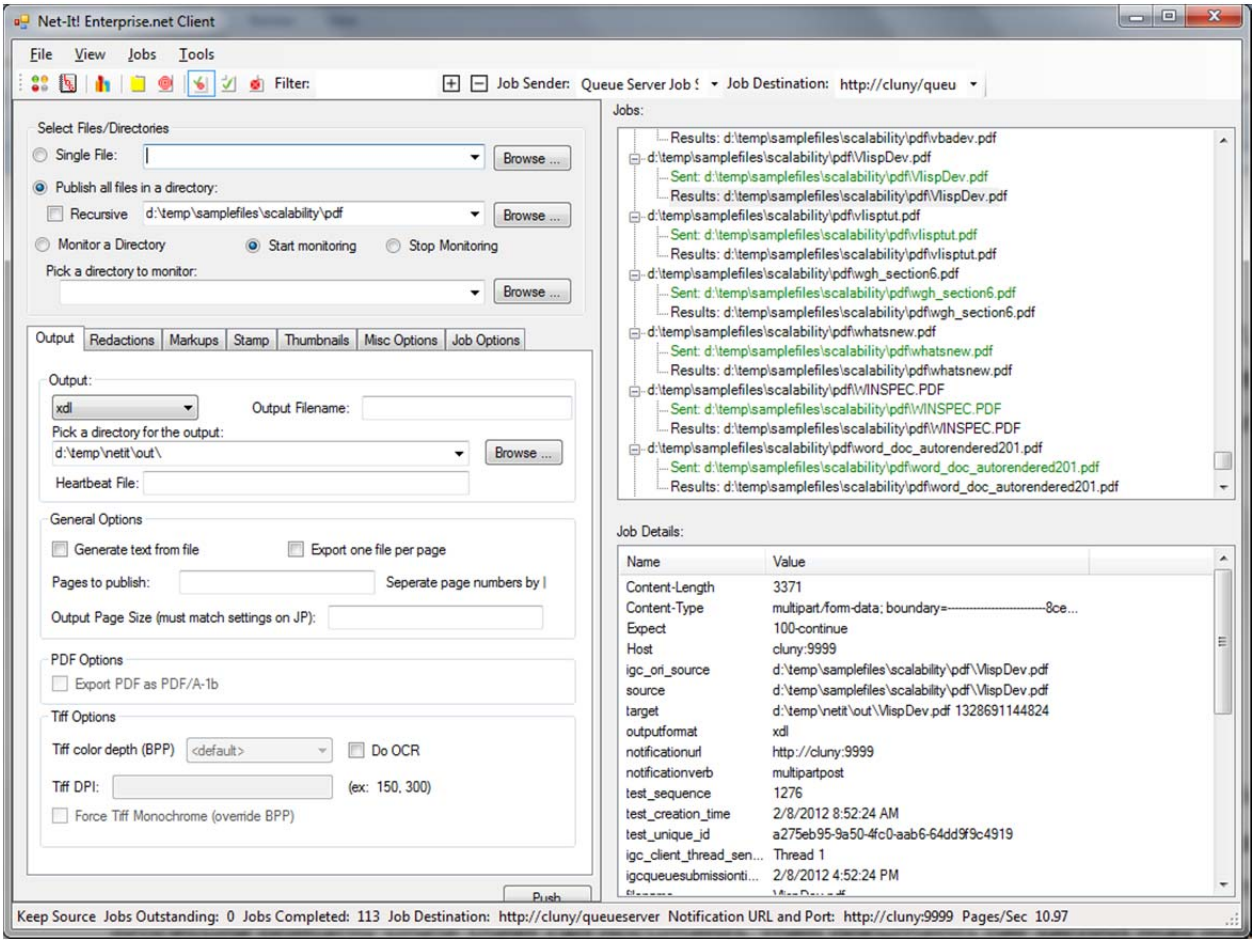
Decide what kind of job you'll be submitting. In this example, we're simply publishing the entire document to XDL. Tests could also include publishing very large JPGs or PNGs (to simulate the flash viewer) or publishing many documents to TIFF (something NIE is good at).

6.3 Configure the JP with a set of job threads

Edit JobProcessor.config and set the thread count to a starting lower limit. In this case, we're testing on an eight-core machine, so we'll start with an initial thread count (on the PDF queue, since we're only testing PDF source files) of four and work up from there. Save the config file and restart the JP.

6.4 Send the jobs

Using the test client, send at least one job for each file.



6.5 Calculate the results

When the jobs are complete, you can see the pages per second on the status bar of the test client. Note that number. Alternatively, you can use the Save Result Jobs as CSV to save the results. We're going to calculate the rate of publication in pages per second, the preferred

measure, since documents per second can be imprecise because of variation in document page count. Megabyte per second also is too vague of a measure, as some documents may be small in size but contain many pages, while others may be large in size but contain few pages. If, however, you're processing primarily single sheet CAD documents, then documents per second may make sense as a measure.

To calculate the pages per second from the CSV file, open it as a spreadsheet, and find the minimum value for the starttime field. Find the maximum value for the endtime field. Subtract the two, and that's the time taken to run all the jobs. Convert that value to seconds, and divide by the total number of pages processed (sum up the publishedpagecount for all jobs). That's your average pages per second.

See Scalability Calculation Speadsheet.xlsx

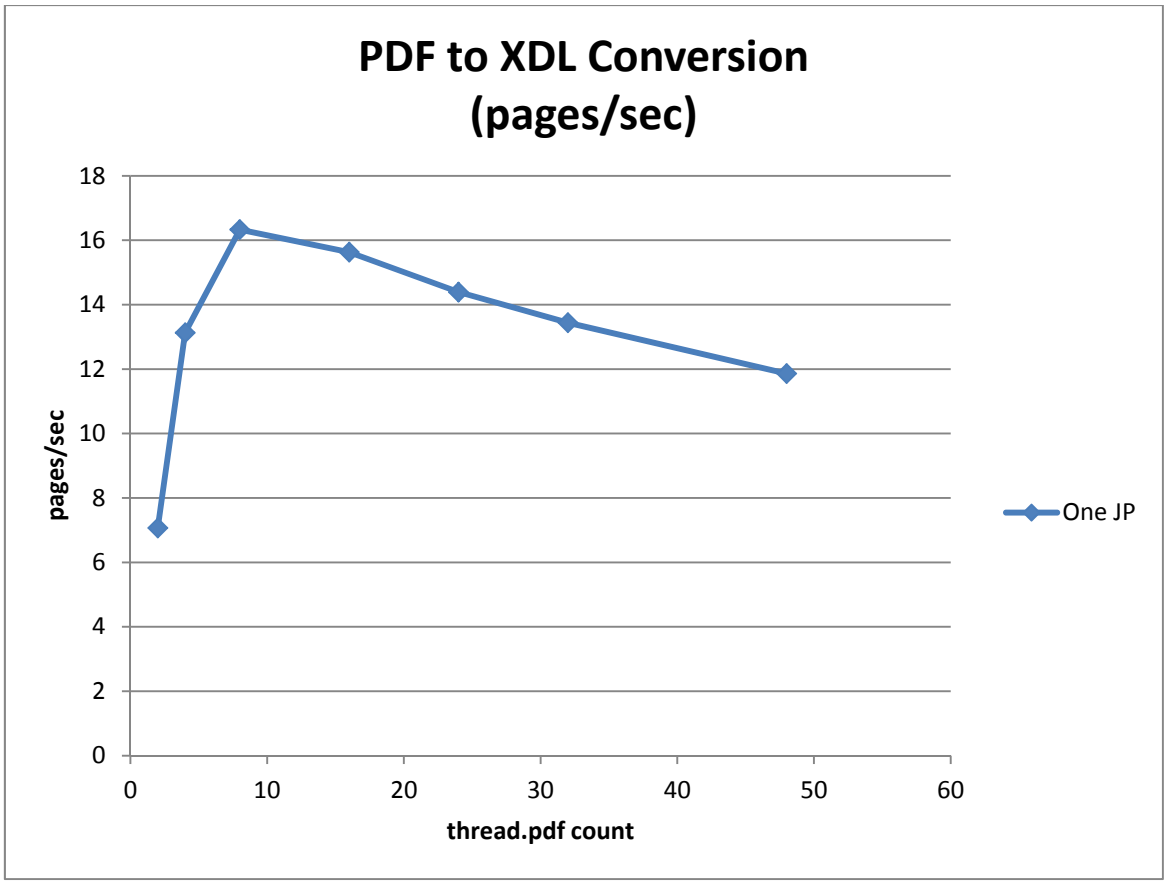
6.6 Reconfigure the JP and repeat

Now, stop the JP, go into the JobProcessor.config file, and increase the thread count for PDF to eight. Restart the JP, clear the jobs from the test client and repeat the test, 5.4 and 5.5 above. Continue to do this as you increase the thread count incrementally.

6.7 Chart the output

After assembling the data from each run (pages/sec versus thread count), you can graph them (pages/sec on the x axis, thread count on the y axis) and see how the values increase until some point and then level out. Set your production environment to use the thread count that corresponds with the highest point on the graph.

Your graph should look something like this:



7 Appendices

7.1 Brava! and Net-It Queues

Brava Enterprise, Net-It Enterprise and Redact-It Enterprise all use the same queuing logic to sort incoming files into the appropriate queue for publication. Job processors are then configured to service one or more queue with one or more threads. This is the fundamental way that the publishing activity is scaled up (on a single JP machine) or out (across multiple JP machines).

Files are sorted into the appropriate queue based on their extension (no file type detection is performed at this point except the actual extension in the filename) in the mapping in the Server.Properties file. The entries look something like this:

```
publish.request.types=doc,drw,pdf
publish.request.extensions.doc=doc,docx,rtf,xls,xlsx,xlw,...
publish.request.extensions.drw=000,906,907,bmp,cal,cg4,...
publish.request.extensions.pdf=pdf
```

`publish.request.types` establishes three queues on the server, called doc, DRW and PDF. Each queue type must have an extension mapping, established in

`publish.request.extensions.<type>`. Each line establishes what extensions go in each queue.

The Brava Server has two additional queues, called "single" and "prq." Single contains single-page publish requests (for those types that support it as established by the `single.page.publish.allowed` setting in server.properties. Prq is used for files whose extensions don't map any extension given in all the `publish.request.extensions.*` entries.

By default, the JP is configured to service all of these queues (all five in the case of Brava JPs, just the three given in Server.properties in the case of NIE and RIE).

Integrators and installers are free to establish additional queue types for added granularity. For instance, if TIFF files are frequently published, perhaps the following line would be added to Server.properties

```
publish.request.extensions.tiff=tif,tiff
```

And the TIFF and TIF extension would be removed from the DRW queue.

If you establish new queue types, you must establish matching settings in at least one JP to service that queue. Modify jobprocessor.config as follows:

```
thread.tiff=3
```

This separation of queues allows you to portion work out to different JPs. A common scenario is to create many JPs that service the .single queue in Brava to provide the best possible responsiveness when a file is viewed for the first time.

For NIE, the several JPs could be established to only publish PDF files, for instance, if a very high volume of PDF jobs are expected.