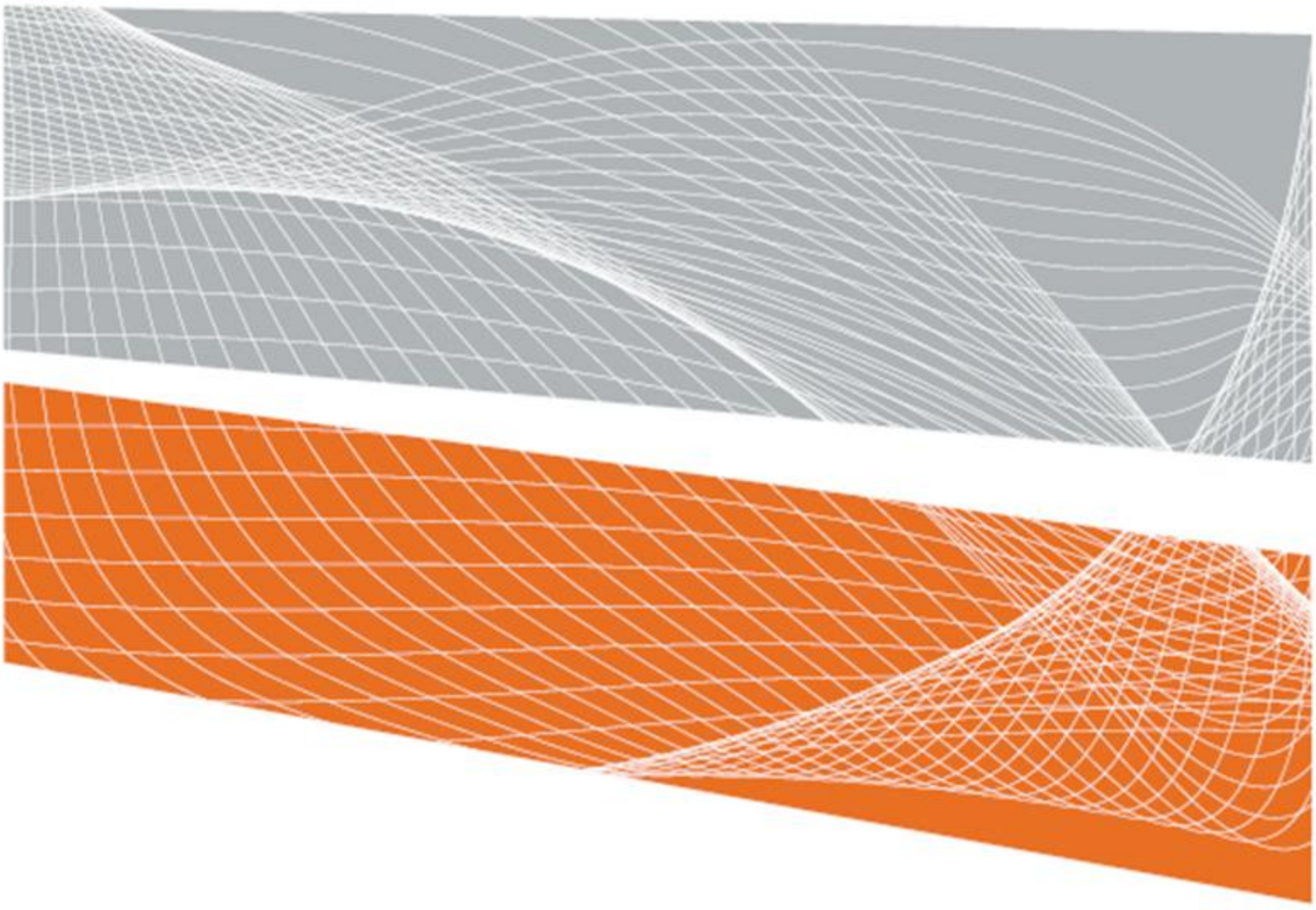

Extend the reach of GIS

Best Practice Guide for Performance

July 2012, Revision 1.1

Contents

1 Introduction

Dekho is an Enterprise Web Mapping Solution. Dekho makes it easy to create and manage Web Maps which extend the reach of Geographic Information Systems (GIS) throughout an organisation and to the public.

This guide provides some practical information for the configuration and implementation of Dekho, exploring common workflows and system configurations. Readers of this document require some knowledge of both ArcGIS Server and Dekho administration

The purpose of this document is to examine the configuration factors that could have a negative impact on system performance with Dekho, and to illustrate best practices that effectively support various business requirements from Dekho. Throughout the document, configuration insights are highlighted in a separate box with the symbol  . Where further detail is available on a specific topic, we have also embedded links to our web site in this document.

This guide is intended to be a living document that will be updated with the product and configuration best practices. Many of details and workarounds listed in this document are the result of direct feedback from Dekho users. As such, any suggestions for additional areas to address in this document are welcomed.

2 A 'Typical' Dekho Deployment

To keep this document limited to a defined scope, all information provided is based upon a 'typical' deployment of Dekho. In reality, no two Dekho implementations (except your Test and Production) look alike. However, it will assist in providing practical information on the support capacity (e.g. number of supported users) for scalability of a 'typical Dekho deployment'.

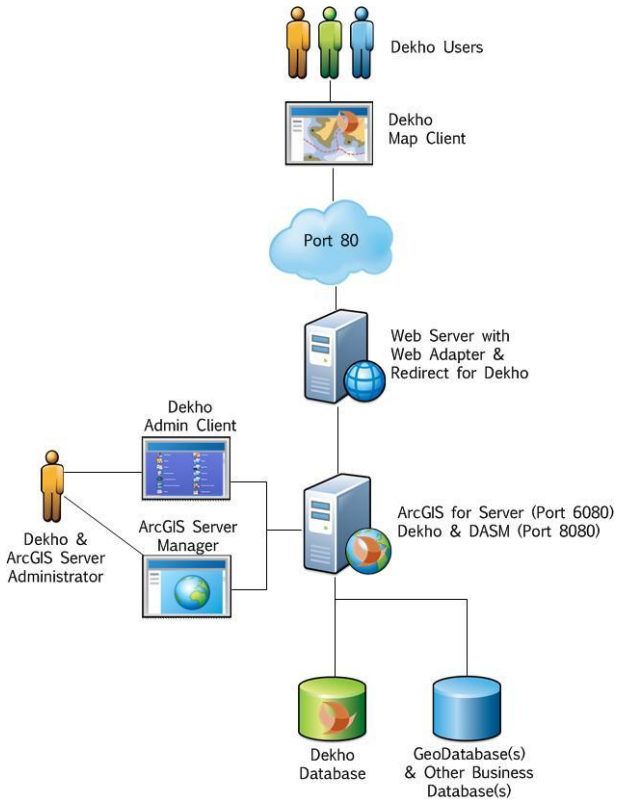

Figure 1 - A 'Typical' deployment for a small to medium scope deployments.

In an organisation requiring a small to medium scope deployment of Dekho and ArcGIS for Server, a typical deployment may resemble a solution stack similar to figure 1 and contain the following tiers:

- ArcGIS Server Site only contains one GIS Server.
  - Dekho and DASM components installed on an Apache Tomcat on the same machine as ArcGIS for Server 10.1
  - A physical 64bit Windows Server 2008 R2 Operating System.
  - 8GB RAM and a 4 core mid-range CPU (~2.8 GHz)
  - Configuration Store and Server directories remain on same machine, but if deployment is to scale to multiple GIS Server machines, these would be on a separate UNC accessible share.
- ArcGIS Web Adapter installed within IIS on a separate web server tier.
- Dekho Database on a separate tier in either Oracle or SQL Server.

It is important to note that this 'typical deployment' is not a reference for best practice, as specific deployments vary based on business and user needs of Dekho and ArcGIS for Server. Please refer to the **Infrastructure Performance Considerations** and **System Design Strategies** wiki for assistance in designing a Server infrastructure that would meet your business requirements.

The next 5 sections of the document describe the 4 key factors which could impact performance in a 'typical' deployment scenario. Those key factors are:

- Authoring Dekho Maps
- Configuring the Dekho Database

- Designing Queries
- Bookmark Management
- Configuring Tomcat Memory

The document will also touch on how variations to this deployment can impact performance.

3 Authoring Dekho Maps

The *Author, Publish, Use* workflow is important to understand with Dekho. If best practice is not followed during the authoring stage it will lead to performance issues when publishing Dekho Maps for users to use.

3.1 Map Service Authoring Considerations

Each Dekho map can be comprised of multiple services (see figure 2). It is the ArcGIS Server administrator/publishers responsibility to ensure these services adhere to best practice before publishing and consuming within Dekho Maps (see figure 3).

If these map services have not been authored with performance in mind, it will result in slow map rendering times which will flow through to the Dekho Map Client.

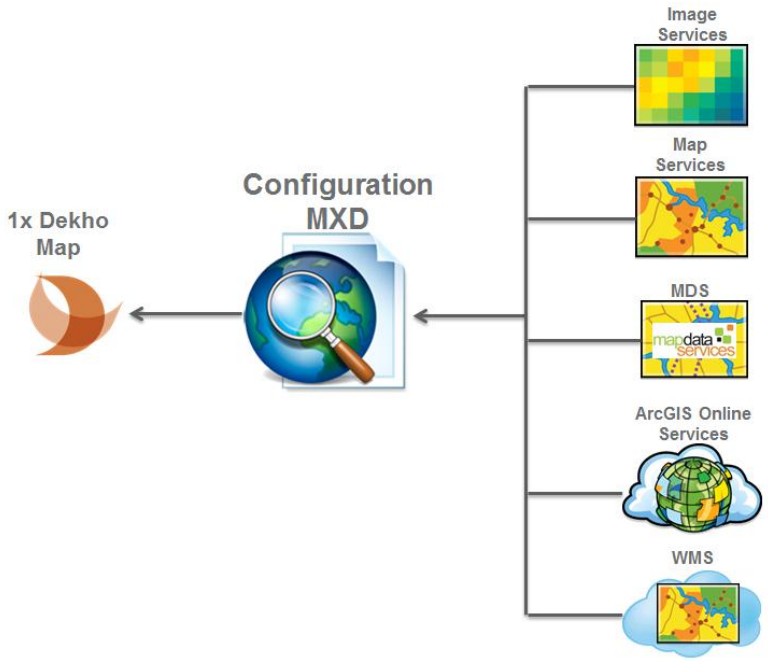

Figure 2 - One Dekho map can be comprised of multiple services.

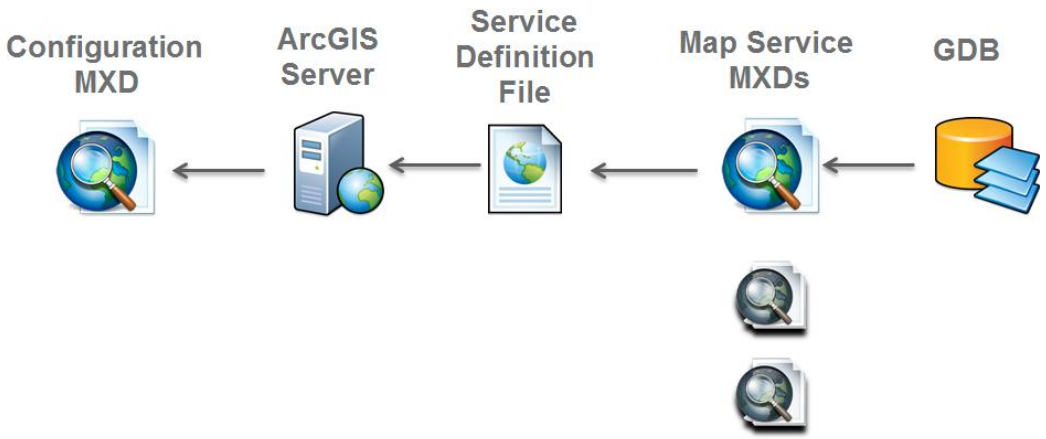

Figure 3 - Ensure map service authoring adheres to best practice before being consumed in Dekho

It is out of the scope of this document to provide a comprehensive list of the do's and don'ts for map service authoring. The key authoring considerations are listed below and for more details you can refer to the **Map Authoring Considerations** in the ArcGIS for Server help. For reference only, the top 8 guidelines are:

- Set scale-dependent rendering for data layers
- Remove unused layers and data frames
- Use definition queries appropriately
- Simplify layer symbology
- Use annotation instead of labels
- Simplify labels
- Set scale-dependent rendering for labels
- Use the same coordinate system for data and maps

Complexity does not scale well with dynamic map services. As the amount of layers, features within layers, or verticies within features increases, rendering times will be impacted. Ensure that only the layers that the users really need to see are displayed. Make use of scale ranges/definition queries; and simplify complex layers for small scales, switching to detailed layers for large scales.

The **service analysers** at 10.1 do address and enforce a certain level of best practice for authoring your map services in ArcGIS for Desktop before publishing to ArcGIS for Server.

When publishing your services, ensure all analyser messages are reviewed and attempt to take actions to resolve these messages.

It is also recommend to make use of the **MXDperfstat tool**. This will report back on response times for multiple scale levels, layer refresh times for each map scale, provide layer performance statistics (such as number of features, vectors, labelling), and breaks out display time for several key rendering phases (such as geography, graphics, cursor, and database).

Tuning your services is also a vital step before consuming them in Dekho. In particular, configuring the minimum/maximum number of instances for each service. Please refer to the **System Design Strategies wiki** for more detailed information on optimising these settings for map service performance.

3.2 Make use of a Cached Basemap

A map cache reduces the number of dynamic display layers (less processing load), and as a result your users will spend less time waiting for maps to display in the Dekho Map Client.

Use cached map services for serving GIS data with ArcGIS Server whenever possible.

A distinction is made between what layers constitute going into the basemap (published as a cached map service), and operational layers which the Dekho User will work with (published as dynamic map services).

If data changes often, use of a cached service can still be employed. The caching tools at 10.1 have been designed for **script cache updates** and can target specific parts of the cache for updates. These scripts can be scheduled to run so the cache remains in sync with the live data layers.

Queries against layers can still be executed in a cached service, but layers cannot be turned on or off within a cached service.

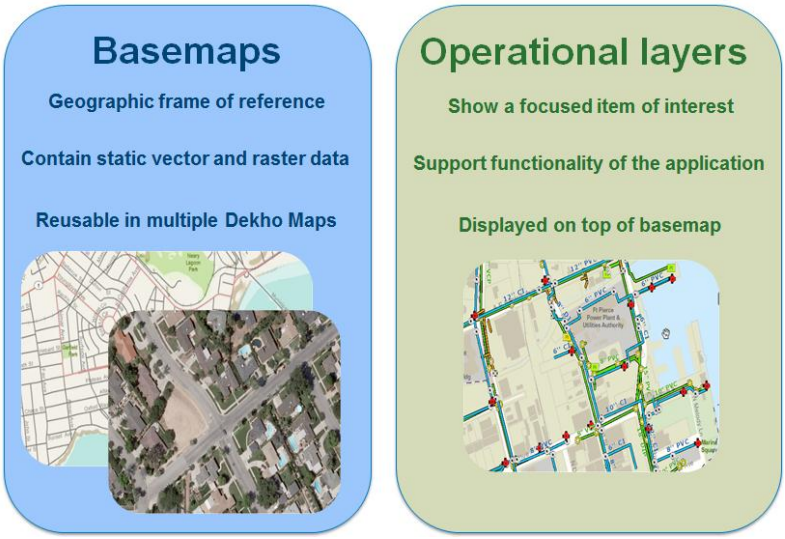

Figure 4 - Distinction between basemaps and operational layers.

It is best practice to make use of a cached basemap for each Dekho Map. Before **building your own basemap** review the **MDS Foundation Map** and **basemaps from ArcGIS Online** for a suitable basemap(s) to use with your operational layers.

3.3 Split Operational Layers Across Multiple Dekho Maps

Although it is possible to publish all operational layers within one dynamic map service and publish alongside a basemap as one Dekho map. It is recommended that the operational layers are separated into logical themes within business data, and published as separate Dekho maps (see figure 4).

This offers two clear advantages:

- Allows for easier allocation of Dekho Maps to specific Dekho Roles that require access to the themed maps.
- Avoid having a bloated dynamic map service containing all operational layers which will have an impact on performance.

If a user needs to print, they can only print layers from one Dekho map and not a combination of layers from different Dekho Maps. Select by selection and measurements between layers in different maps are also not possible. Common layers can be duplicated across the multiple Dekho maps to get around this problem.

With data split across different themed Dekho Maps, users can still toggle between the different Maps and the extent can be retained so that the user retains focus on only their area of interest.

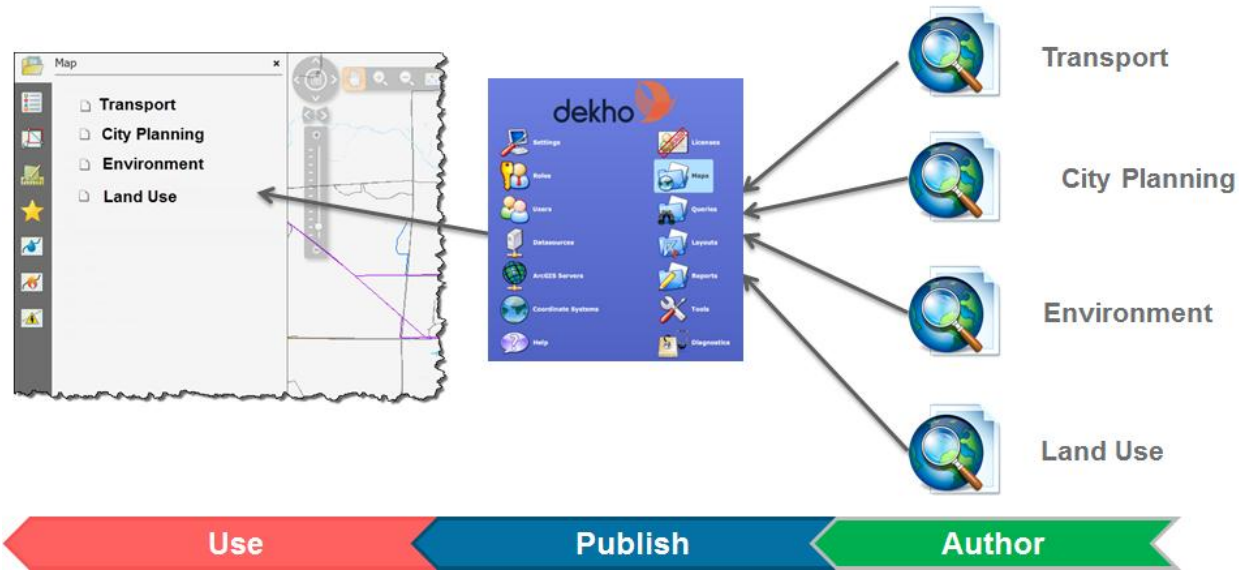

Figure 5 - Split content across multiple themed Dekho Maps.

If business requirements call for a large number of operational layers within a single Dekho Map, along with adhering to best practice for map service authoring, a good approach is to:

- Publish the service with the majority of the lesser-used layers turned off by default. This avoids unnecessary wait times on initialising Dekho.

- Don't spread the layers across multiple dynamic map services within the same Dekho Map, but consolidate them into a single dynamic map service. One request is sent to the GIS Server, one SOC process will draw the map, and one image is sent over the network. This will scale better than spreading the load across multiple dynamic services.

3.4 Layer Description Tags

Layer description tags are applied at both the Configuration MXD and Map Service MXD level. These tags carry behaviour through to the Dekho Map Client and can be used to improve performance for the Dekho Users. A good approach is to follow the below steps:

- To avoid performance issues with the selection and identify tools make use of the *IDENTIFY=FALSE* and *MAP_SELECTABLE=FALSE* tags. Use of these tags avoids layers (particularly basemap layers) from being unnecessarily queried by default. Most users do not typically think to turn the selectability of layers off before making a selection.
- Ensure you apply *TILED=TRUE* to all Cached Map Services within the Configuration MXD.
- If you have multiple basemaps (e.g. different imagery acquisition dates), make use of the *EXCLUSIVE=TRUE* tag (see figure 6). This avoids the Dekho Map client unnecessarily calling multiple basemap services when it should only be calling one.

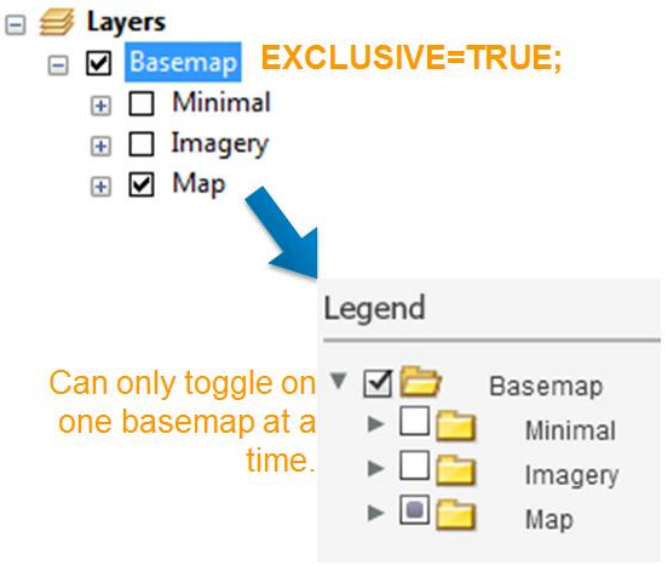

Figure 6 - Allow Dekho users to toggle between basemaps.

4 Dekho Database Configuration

The Dekho application communicates with this database frequently. As users increase, so do the requests between Dekho and this database. If any performance issues exist at the database layer, these will flow through to the Dekho Map Client and degrade performance for your users.

The following sections define the two biggest threats to the performance of a Dekho database installation; and outline recommendations to avoid any impact on performance.

4.1 Network Latency

Referring back to our typical Dekho deployment (see figure 1), the database is located on a separate tier. If there is a significant latency delay between the database and application tiers, this delay will impact the performance of Dekho.

Latency is easy to measure, using a simple **Windows command prompt**:

```
tracert "server host name with database on"
```

Results provide the number of network hops and the associated network latency time for a single trip. High performance LAN should have minimal amounts of latency (~5ms), but if latency times over 1.5 seconds are experienced, then significant delay times for the Dekho Map Client will result.

Unless network latency can be minimised by upgrading the network link between the two tiers, a workaround can be implemented by placing a database instance local to the Dekho Application server (e.g. Oracle or SQL Server Express). This configuration eliminates any latency and network bandwidth issues, but will consume some of the local machine resources. Consequently, the database should be tuned to not consume all of the available CPU/memory from the Dekho/ArcGIS Server application.

4.2 System Resource Contention

If the Dekho database is being housed within a shared infrastructure, response times may degrade when other databases are being heavily hit by separate applications (e.g. asset management systems).

The usage of specific databases can be monitored using a **performance counter** to see if symptoms of slow performance in Dekho can be correlated with large numbers of transactions per second in other databases.

In the first instance, it is recommended to work with your Database Administrator to see if performance can be improved at the database tier by re-allocating resource priorities for the various databases.

As with latency, a workaround would be to install a database instance local to the Dekho application which voids any competing external systems. However, the Dekho instance will be competing against ArcGIS for Server and Dekho for resources, so the server needs sufficient capacity to support this total load and each application tuned accordingly.

5 Designing Queries

When executing queries against external databases, if a query performs badly, it is quite easy to assume the database is the culprit and Dekho is the victim. However, it could be that the Dekho query is the culprit.

Designing efficient queries is important to avoid both wait times for Dekho Users and putting excessing load on the database.

5.1 Make use of Map Service Sourced Queries

Map Service sourced queries are constrained to only pull attribute data from the map services within the Dekho Maps. Compared to JDBC sourced queries, they also lack the ability to alias fields within the SQL or pull in additional tables from non-spatial tables.

If a query requirements are not affected by these constraints, it is recommended to use map service sourced queries over JDBC. Map service sourced queries have quicker response times than JDBC sourced queries.

A JDBC query will first need to find the associated record(s) from the database, and then do a 2nd trip to the map service to pull in the corresponding feature from the geodatabase layer. With a Map Service query, both the attributes and feature are returned in one trip from the geodatabase layer.

5.2 Only Select the Fields Needed

When selecting the fields from the database for a query, avoid using a *SELECT ** and pick only the specific fields that the Dekho User will need to see in relation to that query. More fields will mean longer query execution times.

5.3 Avoid Queries That Return Large Resultsets

If the return of large resultsets is required by Dekho users, it is recommended to use OneSearch queries. OneSearch queries allow for large resultsets to be returned, with no impact on the database, as the information is being drawn from a search index.

For standard search queries, these should be designed to aim to return small resultsets to the client to avoid long wait times for the Dekho User that could lead to potential timeout issues or overloading the database server. It is important to consider the following:

- Make use of multiple input parameters
- Look at making some input parameters mandatory.
- For spatial queries, determine the maximum number of records that can be returned from a query, and ensure this is not too large a result.
- Where possible, pre-calculate as much information into separate attribute fields. e.g. If performing a spatial query to select properties in a postcode, would be quicker to execute if the a spatial join was performed beforehand, and a postcode attribute applied to the buildings layer.

- Avoid increasing the *maximum number of records returned by the server* from its default 1000 value. Increasing it will have a performance hit and put more load on the GIS Server.

5.4 Make use of Database Indexes

Indexes are vital for efficient data access and can improve the speed of SELECT statements. In particular it is important to index any of your fields that you are making a join on.

However, exercise caution and work with your Database Administrator to build indexes, as they could slow the performance of INSERT, UPDATE & DELETE statements.

Please refer to the relevant query performance tuning documents within your DBMS vendor's documentation.

Spatial indexes should also be utilised for faster spatial queries.

5.5 Make use of Database Views

Dekho queries enable access to fields from multiple tables (from the same Data source) and provide the ability to join them to an associated Map Layer.

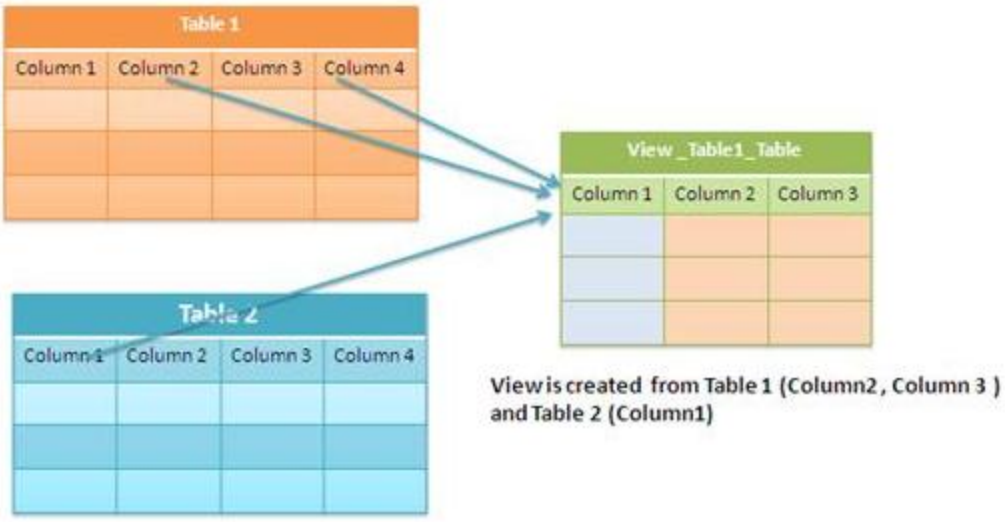

Figure 7 - A view is a 'virtual table' that allows fields from multiple tables to be used.

Building a view at the database level makes it easier to construct queries within Dekho and may also be useful for restricting Dekho to only access a certain subset of data in one or more base tables.

In addition to the added convenience, slight performance gains can be achieved by using views. Again, working with your Database Administrator is recommended to understand where views might be useful for Dekho queries.

If all database performance best practises have been adhered to, performance gains can be achieved by publishing the view (or the results of an SQL query) to a standalone table in a published geodatabase on a regular basis.

5.6 Indexing with oneSearch

Dekho oneSearch queries leverage the Apache Solr search platform and allow Dekho users to utilise pre-built search indexes, created by the Dekho Administrator.

The indexing process is similar to when a cached map service is created with ArcGIS for Server.

- Indexing will utilise memory from Apache Tomcat.
- For map service layers, oneSearch will make batch calls to the service's SOAP handler to build the search index. This will utilise the map service under ArcGIS for Server, and if applicable, also the underlying DBMS where ArcGIS for Server is pulling the information from.
- Indexing JDBC Search Queries will run batch calls direct against the DBMS, using the SQL Statement defined in the original Search Query.

To avoid putting load onto both ArcGIS for Server and the DBMS, it is recommended that the re-indexing of oneSearch queries should be scheduled to occur during off peak hours (see figure 8).

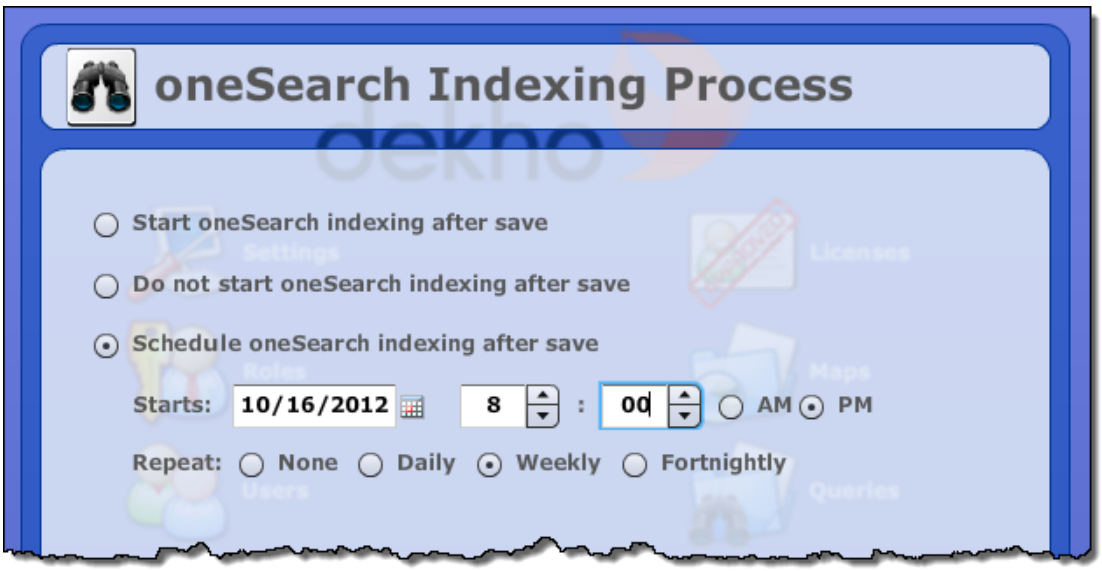

Figure 8 - Make use of the built in scheduler to index oneSearch queries in off peak hours.

6 Bookmark Management

Bookmarks record session state information like, map extent, and map layer visibility, selected map features and redlines.

Dekho Users can save bookmarks to be either private, shared with a role or made available to everyone. If the Dekho Users create a large number of shared bookmarks, this will increase response times from the Dekho database and can cause a delay for Dekho Users opening the bookmark dropdown.

This response time is exaggerated and more noticeable for Dekho Users, when the Dekho Database is hosted on shared infrastructure on a separate tier. Refer to the **Dekho Database Configuration**

section for more information on minimising response times due to network latency and System Resource Contention.

A Dekho Administrator can create bookmarks and then make use of simple bookmark mode from Dekho Settings. This mode prevents Dekho Users from creating bookmarks and restricts them to choosing the bookmarks that the Dekho Administrator provides.

The Product Development Team is currently working on improving the functionality for allowing Dekho Administrator more control over the bookmarks functionality within Dekho.

7 Configuring Tomcat Memory

If performance issues occur with Apache Tomcat, a common cause is a lack of Java Virtual Machine (JVM) memory allocation. The default install of Apache Tomcat can be sufficient for a development environment or a small scale production environment, but as throughput increases with the use of Dekho, additional memory for Tomcat may need to be allocated to meet usage requirements.

Windows Task Manager/Resource Monitor will provide an indication of how much CPU and memory is being used by the Tomcat process. However Java monitoring tools should be implemented to enable more accurate profiling of application performance and memory consumption.

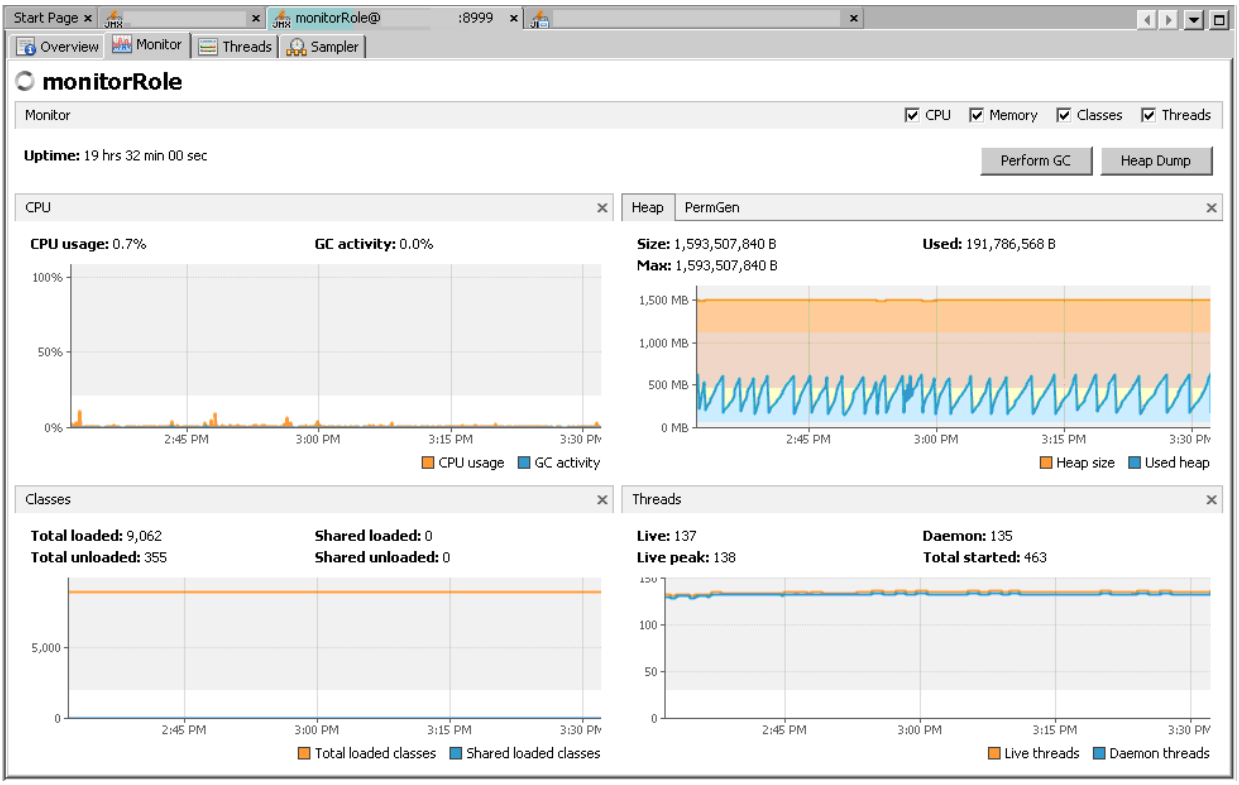

Figure 9 - Real-time monitoring using VisualVM

With monitoring in place, alerts will be generated when memory usage approaches the maximum limit. Such alerts enable action to be taken to increase memory prior to a production environment impact.

For more details on setting up monitoring with VisualVM, please refer to **this post** on our Dekho Resource Center.

If monitoring is not in use, Tomcat could consume memory and begin causing 'Out Of Memory Errors' in the Dekho log files. Dekho users may also report issues such as selections taking longer than usual to select, especially during peak capacity periods.

These errors are generated when there is no space in memory for the JVM to continue working. There are two main causes for this issue:

- Java heap space limitation.
- Java perm gen limitation.

7.1 Configuring the Java Heap

Insufficient Java heap space is the most common cause of the out-of-memory issue. To avoid this issue, heap space settings should be configured using *–Xmx* (Maximum heap size) and *–Xms* (Initial heap size) parameters.

Starting the JVM with a higher heap size will reserve more memory for Tomcat to use. As a result, the JVM won't need to invoke the garbage collection as often and the server will use a higher percent of its CPU time to serving requests for Dekho and ArcGIS for Server.

With Tomcat 7 running as a Windows Service, this is easily configurable via the Tomcat properties GUI, as shown in figure 9.

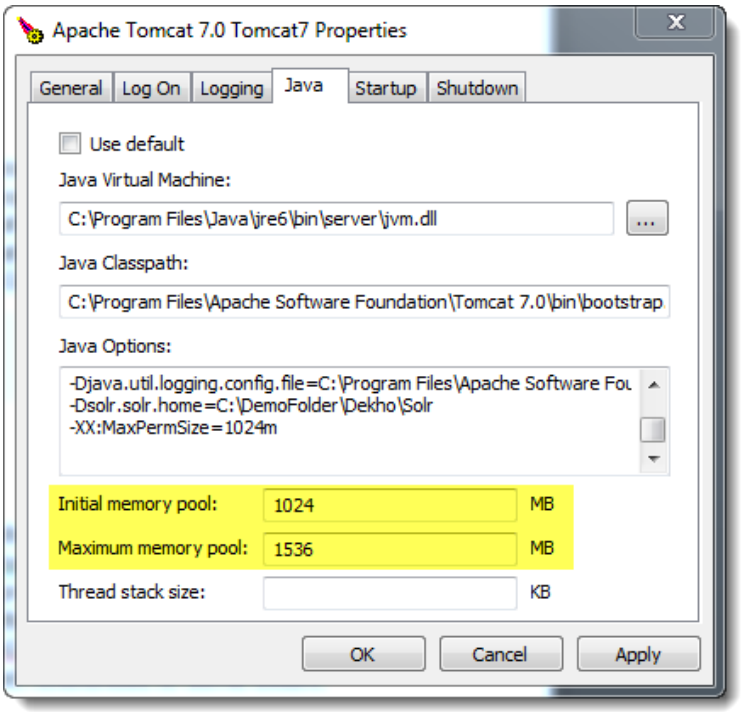

Figure 10 - Setting the *-Xms* (inital memory pool) and *-Xmx* (Max memory pool) for defining the heap size.

The value specified depends on the complexity of the environment, number of users, workflows in Dekho, server infrastructure and what other applications are being used on the same server.

In our typical deployment (see figure 1) 8GB RAM is available and the machine is co-shared with ArcGIS for Server.

For medium sized deployments with around 200 concurrent users accessing Dekho through 'typical' workflows (selections, queries, reporting, etc), a good starting point will be to increase the initial memory pool to 1GB, and the max to 2GB. Ideally this should be verified prior to deployment to production, with optimum memory settings established in a test environment using load simulated by tools (like **JMeter).**

7.2 Configuring the Java Permanent Generation (Permgen)

The 'Permgen' command refers to the location in memory where the JVM stores the class files that have been loaded into memory. This is different and distinct from the heap, which is where the JVM stores the object instances used by an application.

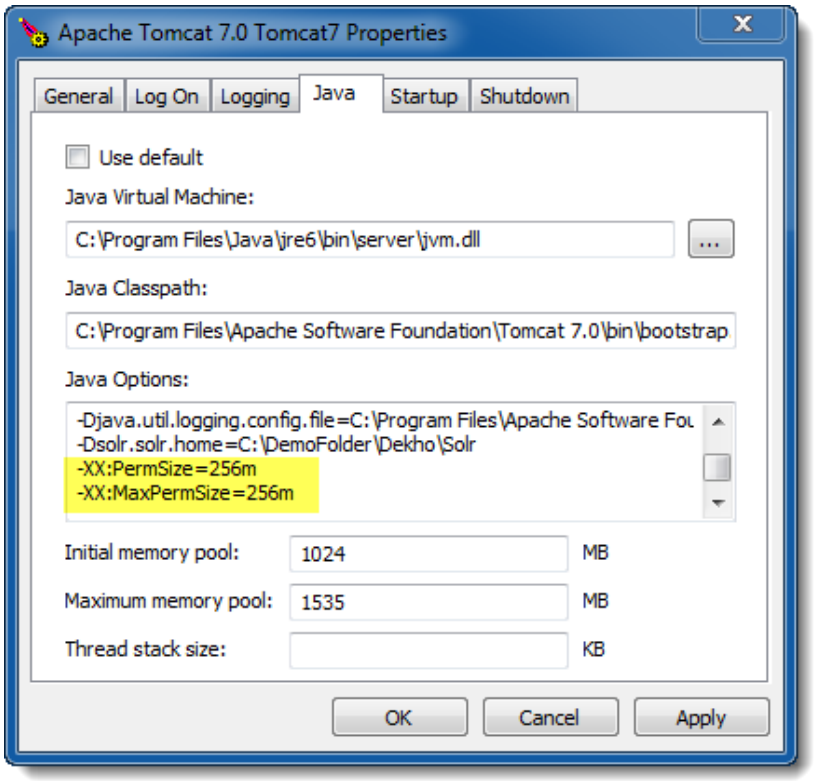

Figure 11 - Setting the permgen using -XX:PermSize and -XX:MaxPermSize

Permgen can be configured using *–XX:PermSize* & *-XX:MaxPermSize* parameters as shown in figure 10.

It is recommended that the initial and maximum values for the size of the permanent generation be set to the same value. This will instruct the JVM to create the permanent generation so that it is initially at its

maximum size and prevent possible full garbage collections from occurring as the permanent generation expands to its maximum size.

On a default install of Tomcat, this setting is 64m and for most deployments this is not sufficient. In our 'typical deployment' with a scenario where around 200 concurrent users are hitting Dekho, a good starting point is to increase the setting to 256m.

For more details on configuring these values, please refer to **a post** on our Dekho Resource Center.

8 Printing

Dekho allows users to export the Dekho Map to variety of formats, making use of ArcObjects *IExportOperator* to produce high quality exports.

Exporting is done under the DekhoAOServer.exe process spawned from the Dekho ArcObjects Server Manager component. Exporting can use a significant amount of memory. Total consumption depends on three main factors, as outlined below:

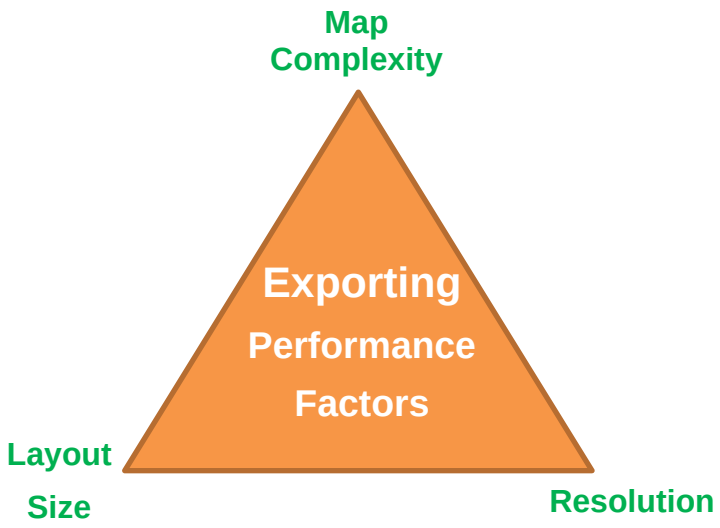

Figure 12 - Factors impacting export performance.

- Map complexity refers to how much data needs to be rendered, how many lables, complex symbology, lots of redlines, etc. As map complexity increases, so does the processing times required to export.
- If the layout size is increased, Dekho has to export a larger total image and this will also increase processing times.
- As the resolution increases, the export will take longer to render requests at higher resolutions, also resulting in longer processing times for exporting.

In Dekho settings, available resolution should be specified by an administrator on behalf of Dekho users. It is recommended to restrict the resolutions down to only those required by the user base, and within what the deployed infrastructure can handle.

To export layout sizes larger than A3 at resolutions higher than 150dpi, or to a large number of users, a more scalable approach is to implement **the custom ArcPy sample** available on the Dekho downloads gallery.

9 Summary

Successful implementation of Dekho requires configuration settings appropriate to your environment and user requirements. Prior to production implementation, careful planning and testing are required to ensure the production environment is not the only test bed.

A good understanding of all components that Dekho relies upon is critical in putting together an implementation strategy for Dekho. Implementing Dekho without this knowledge is a risk that can lead to wasteful spending and poor user productivity.

The purpose of this document is to inform administrators of the various factors affecting Dekho performance, outline guidelines on how to get the most out of your Dekho application, and provide links to greater detail on key subjects.

If more specific information is required or to receive assistance with the configuration and implemention of Dekho specific to your business environment, **Esri Australia's Professional Services** team are always available to assist. They can be engaged any time to help you implement Dekho optimally for your user requirements.