Efficient Data Management & Analysis with Geoprocessing

Jason Pardy
Ken Hartling
Outline

• Recommendations when working with large datasets
• 64bit Background Geoprocessing
• New tools and improvements (10.1)
• Usage tips and best practices
• Sharing Geoprocessing/Analysis workflows (Services and packages)
Large Datasets
What are the large data processing challenges for Geoprocessing?

• ArcGIS permits the storage of very large, complex datasets
  - National & global data
• Large datasets can result in massive amounts of feature overlap
• Users expect to analyze and process large data on local PC’s

• How do we manage processing of large data?
Large Data Processing

• Topology engine is an internal engine that performs overlay processing, dissolve, and more
  - It uses logic called adaptive subdivision processing which was added at 9.2 – known as **Tiling**.

• Tools with this logic include:
  - Clip, Erase, Identity, Intersect, Union, Split, Symmetrical Difference, Update
  - Dissolve, Feature To Line, Feature To Polygon, Polygon To Line
Why we (Esri) subdivide the data?

• Tools perform best when processing is done within your machine’s physical memory (RAM)

• Triggered when data cannot be processed within the available amount of physical memory:
  - A dataset contains a large number of features (collection of features with hundreds of thousands or millions of vertices)
  - Large number of overlapping features
  - Overlap of features is complex
How tiling works – High level look behind the scenes

- Start by trying to load and process all the data in memory.
- If all the data will not fit into memory, we use a Quad Tree approach:
How to get it processed

Best practices and recommendations
New Blog

- Be successful overlaying large, complex datasets in Geoprocessing

Large data processing - ArcGIS 10.1

- Better memory management while an overlay operation runs

- No hard limit to amount of memory
  - Adds considerable scalability in 64bit environment

- All best practices and recommendations should still be followed
Overlay Performance Improvement Example

• Dissolve on ArcGIS Desktop on 32bit Windows machine:
  - 750,000 polygons created from detailed raster image
  - Dissolve field with 4 unique values
  - Output = 9 polygon features with 8,295,940 vertices

10 SP4  10.1  Performance Gain
543s  135s  75%
Overlay Scalability Examples

- Failed to complete in 10
- In 10.1 on Windows 7 64bit, 8GB RAM, 12 GB virtual memory

<table>
<thead>
<tr>
<th>Test</th>
<th>32bit script</th>
<th>32bit ArcMap (LAA)</th>
<th>Server 64bit script</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dissolve</td>
<td>48 min</td>
<td>32 min</td>
<td>32 min</td>
</tr>
<tr>
<td>Dissolve</td>
<td>25 min</td>
<td>18 min</td>
<td>14 min</td>
</tr>
<tr>
<td>Intersect</td>
<td>29 min</td>
<td>34 min</td>
<td>21 min</td>
</tr>
<tr>
<td>Intersect</td>
<td>1 hr 53 min</td>
<td>1 hr 53 min</td>
<td>1 hr 37 min</td>
</tr>
</tbody>
</table>

- **Important:** While there are performance improvements, you cannot say everything is faster at 10.1 – however, scalability has greatly improved.
64bit Background – ArcGIS 10.1

- ArcGIS for Desktop – Background Geoprocessing (64x) (Windows)
- ArcGIS Engine – Background Geoprocessing (64x) (Windows)
  - Separate install

This is **not** a solution which answers performance questions. 64x BG is not always faster, but does scale to provide ability to crunch large data that may have not been possible before.
64bit Background – ArcGIS 10.1

• Data Types not supported:
  - Personal GDB (.mdb)
  - Excel
  - OleDB

• Tools not supported:
  - .NET tools (metadata, mobile)
  - Some GDB administration tools (foreground only)
  - Graphing tools
64bit Background – ArcGIS 10.1

- System must be 64bit (64bit OS)
- Creates two new folders
  - C:\Program Files (x86)\ArcGIS\Desktop10.1\bin64
  - C:\Program Files (x86)\ArcGIS\Desktop10.1\Python64
- Total additional size on disk: **480megs**
- Installer size: ~**140megs**
Running 64bit GP outside ArcGIS

• ArcObjects:
  - IGeoprocessor2::ExecuteAsync()
  - Geoprocessor::ExecuteAsync() (.net assembly)

• Python Scripts:
  - Run using the Python 64-bit installed with BG64
Multiprocessing


- Not recommended for overlay operations
  - Topo engine checks for available memory for the system and takes 60% or more. Since all cores share the same RAM, a large percentage of available RAM is consumed, and so on, eventually failing.

- For server services that do large data processing, one process per node (different machines)
10.1 New Tools & Improvements
New Tools

• 99 new tools in core and extensions
  - 50 + for core

• See Desktop help for complete list in “What’s New for geoprocessing”

New Data Access Cursor

Read records into a list using ‘old’ style cursor:

def cursorDict(inputFC, fieldList):
    rows = ARCPY.SearchCursor(inputFC, "", None, fieldList)
    varNames = fieldList.split(";")
    dataList = []
    for row in rows:
        rowVals = []
        for fieldName in varNames:
            rowVals.append(row.getValue(fieldName))
        dataList.append(rowVals)
New Data Access Cursor

Read records into a list using DA cursor:

def daCursorDict(inputFC, fieldList):
    varNames = fieldList.split(";")
    rows = DA.SearchCursor(inputFC, varNames)
    dataList = []
    for row in rows:
        dataList.append(row)
    del rows
    return dataList
### New Data Access Cursor

DA cursor is much faster!

<table>
<thead>
<tr>
<th>Number of records</th>
<th>Cursor</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>DA Cursor</td>
<td>0.087290</td>
</tr>
<tr>
<td>10000</td>
<td>Cursor</td>
<td>15.289117</td>
</tr>
<tr>
<td>50000</td>
<td>DA Cursor</td>
<td>0.447475</td>
</tr>
<tr>
<td>50000</td>
<td>Cursor</td>
<td>76.312171</td>
</tr>
</tbody>
</table>
Array instead of List?

Compare sending record info to a list vs an Array:

def daCursorArray(inputFC, fieldList, N):
    varNames = fieldList.split(";")
    rows = DA.SearchCursor(inputFC, varNames)
    dataList = NUM.empty(((N,len(varNames)), float)
    c = 0
    for row in rows:
        dataList[c] = row
        c += 1
    del rows
    return dataList
def daTable2Array(inputFC, fieldList):
    varNames = fieldList.split(";")
    data = DA.TableToNumPyArray(inputFC, varNames)
    return data
Array instead of List?

Pretty fast and use a lot less memory.

<table>
<thead>
<tr>
<th>Records</th>
<th>Cursor, output type</th>
<th>Time (s)</th>
<th>Memory(kb)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50000</td>
<td>DA Table, Array</td>
<td>0.420963</td>
<td>904</td>
</tr>
<tr>
<td>50000</td>
<td>DA Cursor, List</td>
<td>0.447475</td>
<td>3813892</td>
</tr>
<tr>
<td>50000</td>
<td>DA Cursor, Array</td>
<td>1.768081</td>
<td>816</td>
</tr>
<tr>
<td>50000</td>
<td>Cursor, List</td>
<td>76.312171</td>
<td>5225096</td>
</tr>
</tbody>
</table>
Clip Performance

- Fast when inputs are not M or Z aware, have no curves, and the input XY tolerance parameter is not set
- Triggered automatically and uses the geometry library (not topo engine)

MEGA Clip Test

Input Data: Shapefiles - Input Feature Class: 50000 (high density contour lines)
Clip Feature Class: 1 polygon
Output: 5301 lines

<table>
<thead>
<tr>
<th>10.0SP4</th>
<th>10.1</th>
<th>10.1 (Topo Eng.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failed (after 12 hrs)</td>
<td>34 secs.</td>
<td>10 min 24s</td>
</tr>
</tbody>
</table>
Buffer

• Creates true geodesic buffers for point, line and polygon data
  - geodesic buffers truly are more accurate than Euclidean
  - geodesic buffers will take more time than generating Euclidean buffers
Spatial Join

- New match options – equivalent to the options with Select Layer by Location tool
- Better performance and scalability
  - huge memory leak (Oops!) fixed
Dissolve

• Size of output feature determined by size of available memory
  - Tile boundaries may be completely removed because of this

• Biggest challenge for dissolve is Buffering of points and dissolving.
  - These kinds of dissolve have shown to now complete on 64bit systems

• **Note:** A feature on one system may not draw or process on another system with less memory
Proximity Tools

- Near – intermediate size data:
  - Use to take 3 hrs 37 mins
  - Now takes: 14 secs

- Generate Near Table:
  - Memory leak minimized – used to leak 8MB for 200 iterations (40kb/iter). Now, less than 4KB
Tabulate Intersection & Polygon Neighbors

• New tools
  - Used to be possible by combining numerous tools
  - Faster as a system tool because there is no intermediate data
Make Query Layer

• New a tool at 10.1
• If the result of the SQL query entered returns a spatial column, the output will be a feature layer, otherwise it’s a tableview
  - Results can be used as input to tools
  - Results are read-only
• Performance really depends on the query design and database optimization
Truncate – New at 10.1

- Removes all rows from a database table or feature class using truncate procedures in the database
  - Replaces Delete Features/Delete Rows which did this
  - Fast delete of all records
  - Non transactional
  - Versioned data is not supported - data must be unregistered as versioned before the tool will execute successfully
Feature Class to Feature Class and Table to Table

- Both the subtype and domain codes and descriptions can be included in the output shapefile
  
  - By default, only domain and subtype codes will be included in the output, not descriptions
  
  - Use the “Transfer field domain descriptions” geoprocessing environment to control this behavior
Usage Tips and Tricks
**Situation:** you want to generate a single buffer of all points

**Problem:** this is only 1000 points and it takes 2 minutes to buffer.... this isn’t a lot of data, why so slow?
Why so slow?

- The 1000 input points and output area, looks simple enough
Dissolve is the problem

- Running buffer with the Dissolve option set to NONE takes 1 second… what’s the difference?
Dissolve has too much work to do

- Individual point buffers with only boundary drawn
...as you can see when you zoom in

- The complexity internal to the area is what’s bogging dissolve down
Solution

• Get rid of internal complexity: use the **Aggregate Points** tool (new at 10.0)
  - *Creates polygon features around clusters of proximate point features*
• Run buffer on polygon result of aggregate points
Aggregate Points approach...

- With 1000 points buffer with dissolve took nearly **two minutes**, the *Aggregate Points* approach took a **few seconds**.

- With 5000 points within the same area buffer with dissolve took over an **hour and a half**, the *Aggregate Points* approach still took only a **few seconds**.

  - the amount of complexity grows logarithmic
Use “in_memory” workspace

- Useful for small to intermediate datasets
- Can improve performance of models/scripts
- Be careful that the size of your in_memory feature classes do not put stress on the amount of available memory for processing.
Do selections in-memory

- Use Make Feature Layer or Make Table View tools
  - Select by Attribute
  - Select by Location
  - Add Join

- Instead of Select or Table Select which create a new dataset

- 10.1 - all tools are faster with selection sets
Use Field Info to limit and rename fields in output

- With Make Feature Layer and Make Table View:
  - Set new field name
  - Set a field to be visible or hidden
  - A split ratio can be set

- In output data:
  - Fields will be renamed
  - Non-visible fields will be omitted
  - The attributes of the output features are a ratio of the original feature’s value based on how the geometry is divided
    - i.e. Clip, Split, Intersect
Select Layer by Location

- Use this tool to answer spatial relationship questions
- Fast and scales extremely well
- Uses layer as input

- Can be use to split up large data
  - i.e. Tweet data (massive amounts of points) by continent
Relational and Topological Operators

- 10.0 – Relational operators in Python
- 10.1 – Topological and proximity operators
Split

- Split tool may be a better option than multiple clips
- Features are read once and processed once
  - Faster than calling multiple clips

- BUG IN 10.1 with FGDB – slowed down for large data, fixed in SP1
Extent environment

- Can filter what features or raster data is used for processing
  - Features that pass through the extent are included
  - For raster data, results from running tools will be contained with the extent
Projections

- Process data with the same projection - avoid projection on the fly
- Use the Default Resolution and Tolerance for your Spatial Reference whenever possible.
Joins

- Index your join fields (if possible)

- Join Field tool
  - Adds field(s) from one table to another, based on attribute relationship
  - Updates input table
  - Avoid using add join, calculate field, remove join or add join, copy features, delete fields
Spatial Analyst
Optimizing Spatial Analyst Performance

- “in_memory” workspace now supported
- Native I/O (10.0) – no conversion
  - Trouble with FGDB with tools that have Random I/O i.e. Cost Distance, Cost Path
- Arcpy behavior – There is conversion with file based formats (grid, tiff, etc.)
- Raster Calculator – optimized for arithmetic expression 
  \( (a + b + c) \)
  - don’t use tools in Raster Calculator
- Use Raster Calculator in model builder or stand-alone. In Python, write the expression
Optimizing Spatial Analyst Improvements

- SA tools work better with Image services – statistics calculated on the fly
  - Requirement to use the extent environment
  - Combination of cell size and extent larger than the allowable limit of the server, tool will fail.
Sharing
Geoprocessing Packages

- A convenient way to share geoprocessing workflows
- Created from one or more geoprocessing results in the Results window
- All the data and tools used to create the result are included in the package
- A .gpk can help with:
  - Sharing methodologies
  - Project collaboration
  - Consolidation of projects
  - Training
  - Troubleshooting
Geoprocessing Services

• A geoprocessing task that takes data captured in a web application, processes it, and returns meaningful and useful output in the form of features, maps, reports, and files

• Services can be used by many client applications:
  - ArcGIS for Desktop
  - ArcGIS for Engine
  - ArcGIS for Explorer
  - REST
    - JavaScript
    - FLEX
    - Silverlight
Geoprocessing Services

1. Knowledge about using geoprocessing tools and environments is important for knowing how to create a good geoprocessing service.

2. Knowing and understanding the input data required for the service is important.
Geoprocessing Service Behavior

- Before authoring or publishing, identify what you want your service to do and how you want it to behave with clients:
  - Does input data come from the client or select it from the server?
  - Draw the results with map server or download and draw data on the client?
  - Save data on the server?
How to create a service

• Changed at 10.1 – easier

• All services start from a successful result
  - The result acts as a template to build the service

• Quick tour of Publishing:
  http://esriurl.com/gpSrvQuick
Data Store

• New concept at 10.1
  - Either an enterprise database or a folder
  - All data can live here where all servers have access to this data
    - Can be local or unc path
Data store

• Must specify data store folder
  - LAS dataset

• If these exist in the data store, analyzer will generate an error. You must convert to file geodatabase:
  - Access
  - Coverage
  - Coverage feature class
  - INFO table
  - Excel table
10.1 Geoprocessing Web API

- Faster REST handler
- Uploads Operation
  - Item id supported for file and raster input
  - `{"itemID":"<itemid>"}`
- Cancel Operation
10.1 Geoprocessing

- in_memory raster
- 255 path limit for grids in spatial analyst
- Reduce memory leaks
Geoprocessing & Python Resource Centers