Maintaining High Performance with Big Datasets *and* Interactivity

By Derek Swingley, ArcGIS API for JavaScript Development Team

Dealing with big datasets is a frequent topic of discussion among web mapping app developers. The prevalence of huge, crowdsourced datasets ensures that big data will continue to be a common point of discussion when you get more than two web mapping geeks in the same room. The traditional approaches of generating a raster tile cache or using a dynamic map service work, but if you want any interactivity with the underlying data, it requires a server round trip. This article will discuss some of the key concepts behind building a web app that provides an extensive interactive experience while maintaining responsiveness and performance.

What makes big data tricky is that browsers cannot elegantly handle upwards of a few thousand features. Send more than a few thousand points or a few hundred polygons to a browser and you can watch it grind to a halt, especially if you're running a legacy version of Internet Explorer. This is a problem, because slow performance is a death knell for web apps. If someone using your web app has to wait 10 seconds, 20 seconds, or longer for five megs of JSON data to be retrieved and displayed, there's a good chance they'll just give up and move on. What good is a web app that no one uses?

At some point, you cross a threshold where it's not practical to send all your data across the wire and let the client sort it out. While that threshold is an ambiguous one, you know when you hit it. If you're a JavaScript developer, how should you deal with big data? The answer: feature layers.

I've put together an application, High Performance Maps with Feature Layers (servicesbeta.esri.com/demos/high-perf-feature-layers), that demonstrates how to use feature layers to set up custom scale dependencies and follow best practices to ensure that performance does not suffer when querying, retrieving, and displaying large datasets in an ArcGIS API for JavaScript application.

The goal was to display appropriate US boundaries (states, counties, or census block groups) for the current map scale. For example, trying to display census block groups for the whole United States would overwhelm the browser with features. This app makes sure that only the states appear at the small scales, counties at the medium scales, and census block groups at the large scales. This ensures that a manageable number of features are always being transferred and displayed.

The app is simple, but the concepts can be applied to just about any web mapping app that has to deal with big data. The total size of the data used by the app is around a couple of hundred megabytes. This is accomplished through custom scale dependencies. This is as simple as creating a few feature layers, listening for their onLoad event, and setting their minScale and maxScale properties. Listing 1 is a simple code sample that shows how this is done:

```
var fl = new esri.layers.FeatureLayer(url, options);
dojo.connect(fl, 'onLoad', function() {
  fl.minScale = minScale;
  fl.maxScale = maxScale;
});
```

↑ Listing 1: Custom scale dependencies

The app includes a couple of other bells and whistles that demonstrate some tools that are available when you have your features on the client side. The first is a rich pop-up that displays feature attributes, as well as an option to zoom to the feature when you touch a graphic. The second is the ability to filter features based on population using a slider. Neither tool would perform nearly as well if features weren't available client side.

The final point is that the app's performance is kept snappy because only the required data is being sent to the client. For attributes, only the fields required by the pop-up are sent across the wire. For geometries, that means using maxAllowableOffset to generalize geometries on the server before they're sent to the client.

When you create a feature layer, one of the options is maxAllowableOffset. You might recognize this parameter name from the ArcGIS for Desktop generalization tools, where it's been used for years. If this is new terminology, see the ArcGIS for Desktop Help for the Generalize tool. A bonus: FeatureLayer comes with a setter method, setMaxAllowableOffset, that you can use to simplify features on the fly. Since the web APIs fire an event when the map's extent/zoom level changes, you can listen for this event and use setMaxAllowableOffset to indicate an appropriate value.

Now all you have to do is figure out a value for maxAllowableOffset. Let me suggest a method: A feature's geometry should not display more than one vertex per pixel. After all, a pixel is the smallest unit for our displays, so displaying more than one vertex per pixel is wasted effort. To display no more than one vertex per pixel, we can take the map's width in coordinate space divided by the map's width in pixels, and we have a reasonable value for maxAllowableOffset. With this method, you are telling your feature layer to make sure simplified features do not differ from the original by more than the width of a pixel.

Finally, one caveat. You can't specify maxAllowableOffset on layers that you allow users to edit. If you're editing a feature, you need to see the true representation of that feature, including all its participating vertices. Editing a generalized feature and then sending it back to the server could wipe out detailed information that was meticulously created and maintained, so you can't specify maxAllowableOffset on feature layers that are editable.