Article
Daniel Kutac · Aug 21, 2020 3m read

Open Exchange

Dynamically changing production items poolsize

Dynamic PoolSize (DPS) Experiment

Purpose:

Enhance Ensemble or IRIS production so it can dynamically allocate pool size for adapter-based components based on their utilization.

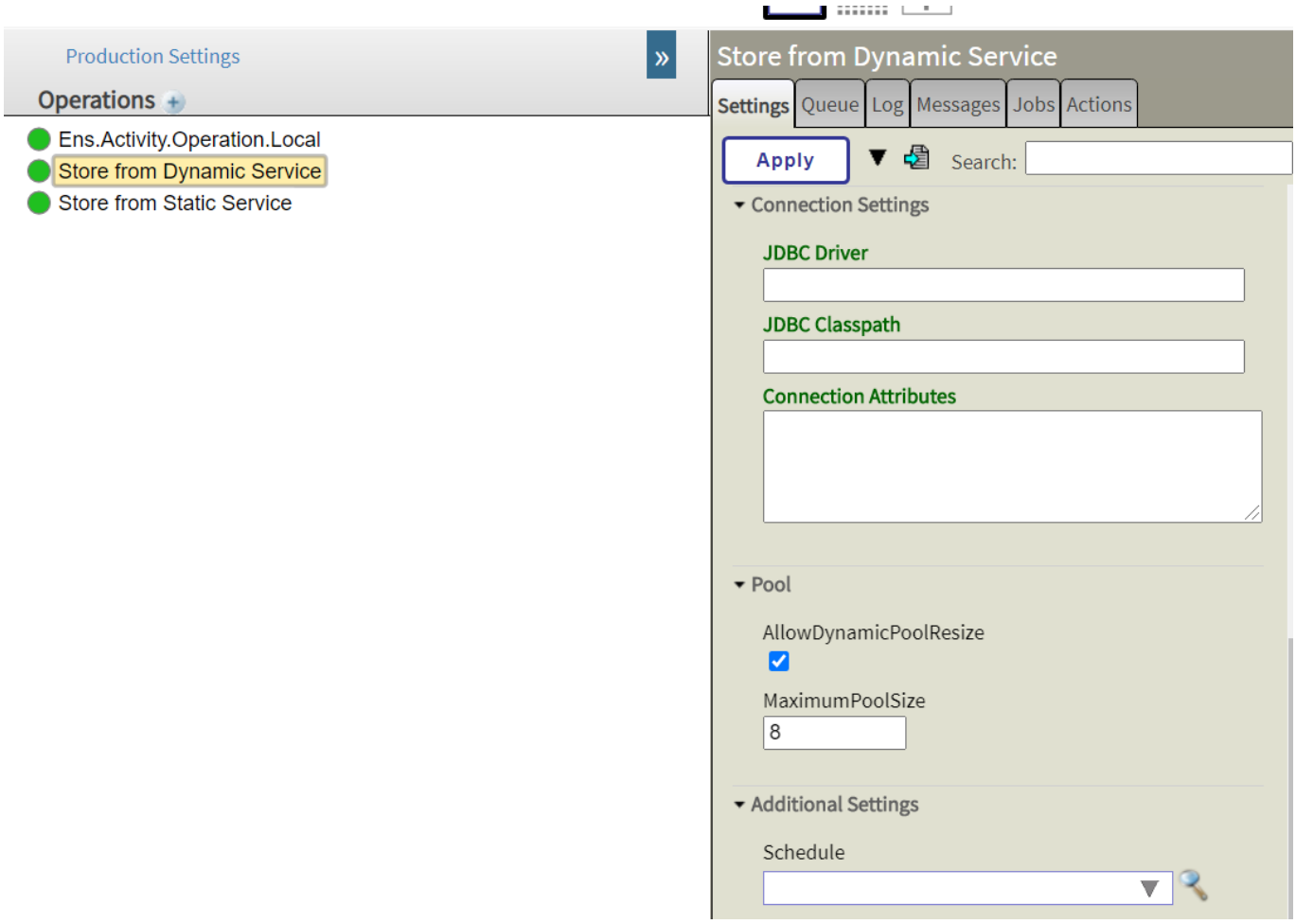
Sometimes, an unexpected traffic volume occurs, and default pool size allocated to production components may become a bottleneck. To avoid such situations, I created a demonstrator project some 2 years ago to see, whether it would be possible and feasible to modify production, so it allowed for dynamically modifying its components per their load.

Implementation description:

Create class **Ext.DynaPool.PoolManager.Adapter** class and inject it to **Ens.Adapter** (modify and compile Ens.Adapter class) so it adds new properties used by the DPS infrastructure. These properties will be configurable in the production settings. Make sure all adapter classes are recompiled in order to benefit from the new feature.

Properties:

- AllowDynamicPoolResize (Boolean) – this enables dynamic pool size allocation
- MaximumPoolSize (Integer) – this limits maximum pool size allowed to avoid oversizing and killing server performance due too many concurrent jobs



Create a new component (service) – **Ext.DynaPool.PoolManager.Service**. This service scans periodically all production items and for those that are adapter based and have enabled *AllowDynamicPoolResize* performs queue length check.

In future, OS related data could be collected too, like CPU usage, memory usage as these data could be used to determine pool size for a given load.

All retrieved data are stored in a persistent classes - **Ext.DynaPool.PoolManager.Storage.Snapshot** – for storing current values and **Ext.DynaPool.PoolManager.Storage.History** – for storing historical data.

Add **Ens.ProductionMonitorService** (this is a service that comes with Ensemble installation) to the production.

The **Ext.DynaPool.PoolManager.Service** class contains method *NeedPoolResize()* that is a placeholder for implementing code that determines pool sizes of applicable components. Currently, an empirical formula was implemented but depending on site characteristics a new formula needs to be developed that would use collected information and determine which of production items need to resize the pool size. For all these items, a new pool size will be set. This service also performs production update when necessary to start new jobs or terminate unnecessary jobs depending on new pool sizes.

Testing:

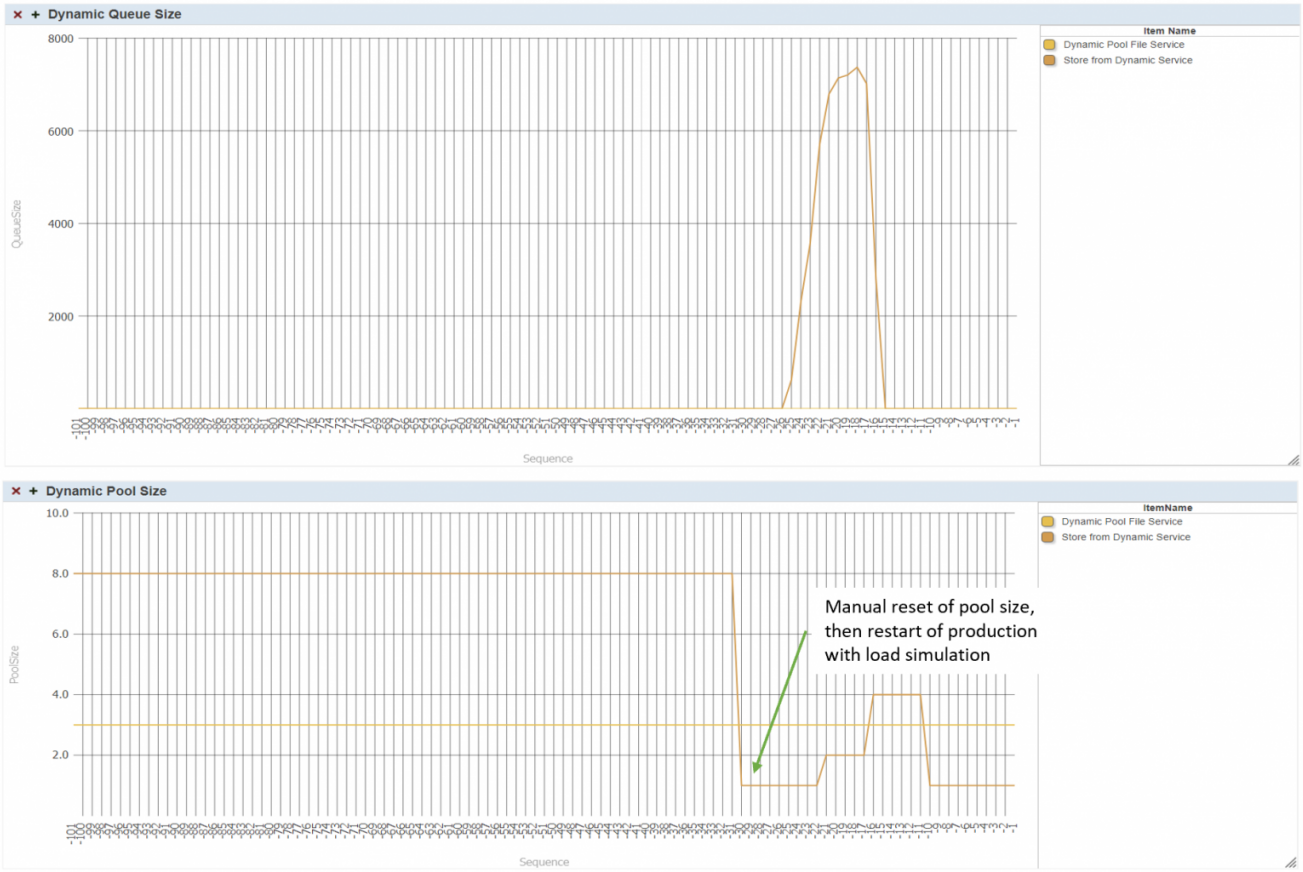
Start production and open terminal. From terminal run this command:

d ##class(DPS.Test.Util.Generator).Generate(10)

This command populates several files with random strings. A test production is provided with project that consists of two file services consuming incoming files and sending them to operation for storing data into database. One service (**DPS.Test.Service.FileService**) is configured to use dynamic pool size allocation and the other uses one static pool size. There are two operations (**DPS.Test.Operation.Database.Store**), and again, one is using also static pool size whilst the other one uses static pool size.

A complimentary Analytics Dashboard is provided that shows both queue length and pool size for each component with dynamic pool size allocation.

Pict. Dashboard



Please note – the code is written for Windows environment, modify it accordingly when testing in other operating system.

Considerations:

- Make sure resizing is not happening too often as this involves component interruption and adds overhead on its own.

Disclaimer:

This code is just an experiment, it has never been tested in real productions and in its current implementation is not intended for production deployment! But perhaps it would serve as a source of ideas (good or bad).

There is also a hack used in **Ext.DynaPool.PoolManager.Service** class that might deserve refactoring. On the other hand, there is no harm to production data as it only deals with statistics historical data.

You can find project files in the Open Exchange.

Enjoy!

#Interoperability #Performance #Ensemble #InterSystems IRIS
Check the related application on InterSystems Open Exchange