Performance Improvement and Outcrossing and installation on old server

Goal:

1. Read the 7 jar files and try to find possible improvements and testing.
2. Read the 7 jar files and study how to add the outcrossing smoothly without affection of the other output files and testing.
3. Install the repaired jar files and restart the services to install the repaired services and testing.

All the source code is in “C:\java\_workspace\kml” on the new server.

**Improvements:**

The original code parses the input file line by line and character by character which is a very inefficient way to read data. The new code replaces the old code with the Java built-in String function “split” and used with regular expression “[\\s+](file:///\\s+)”. The benefit of this method is that a whole chunk of codes is replaced by a single line, which greatly enhances the readability of the code. And it also reduces the time to process the lines. This occurs in many places of the original code in the 7 jar files which takes a lot of time to fix and the new code should be ensured to be able to replace the original code smoothly. This also takes time. One example is listed below:

**if**(eof\_flag\_temp == **false** && read\_templine == **true**){

templine = tempreader.readLine();

**if**(templine == **null**){

eof\_flag\_temp = **true**;

}

**else**{

***str\_tempdata = templine.split("\\s+");***

}

}

Another improvement is that the original code use “Thread.sleep(3000)” to sleep 3 seconds while executing some external programs. Actually, the external programs don’t necessarily spend so much time executing. We’d better proceed immediately after the external program finishes executing, rather than spend a fixed amount of time waiting. There is a method “waitfor” in the Java “process” class which can achieve this. The code snippet is listed below:

Runtime rt = Runtime.*getRuntime*();

Process p = rt.exec("hycs\_std.exe");

//Thread.sleep(3000);

p.waitFor();

p = rt.exec("con2asc.exe cdump");

//Thread.sleep(3000);

p.waitFor();

There are still a lot of places which can be improved. But the above two are the major ones.

**Adding outcrossing:**

In order to add outcrossing, many things have to be done:

We have to add a column in “output.txt”; this is done in “sendemail.jar”.

After adding a column in the “output.txt”, the length of the array in “depositionlegend.jar”, ”generatedepositionkml.jar”, “concentrationlegend.jar” and “generateconcentrationkml.jar” cannot be fixed as 6 when reading the “dep\_conc\_output.txt” file, since we have added another column. Thus this involves a lot of modifications. This can be achieved using the above Java method “String.split([\\s+)](\\\\s+))”. After outcrossing legend and kml file are generated, we have to modify the “zipthefiles.jar” file to zip them. Thus in total all the above work involves 7 jar files. One inefficiency is that the output file is read many times. Actually we can record the range of various variables when we first read them.

**Installation:**

After repairing the bunch of “jar” files, we have to install them onto the old servers and test whether they run smoothly. This involves modifying the task scheduler and testing whether the installation is good. One of the problems occurs in the old server but not the new one is that when you end one task, you cannot end the child processes it spawns. If you try to end the child processes by hand, since there are two “php.exe” processes, you don’t know which one to end. One solution not so decent is ending all the php child processes. And then restart all the services. This is not good, because when you want to just install one php service, you have to stop all the php services first and then restart them. I have tried to find some information on the internet to seek solution to this problem, but no success. Therefore, installation of either “horseweed” or “switchgrass” will affect the running of the other service. After one of them is reinstalled, both of them have to be tested to ensure they run smoothly.