**Using Parallel to copy files to S3**

**Introduction**

If you started to work with AWS services and need to transfer your files to S3 buckets and your company has large directories with thousands or millions files, this article can be useful to you.

**A simple scenario**

**Source:** A folder “CompanyProfileXml” that contains   
 17.917 Sub Folders and 45.090 Files in a total of 181Mb.  
  
**Target:** Amazon S3 – bucket “sindiconet-company-private”

**Using a simples foreach**  
**Code:**  
  
 static void ReplicationFilesRecursive(string localDir, BAmazonS3 pBAmazonS3, string cleanPath = null)

{

foreach (string dirPath in Directory.GetDirectories(localDir))

{

string currentFolder = Path.GetFileName(dirPath);  
string currentKey = cleanPath != null ? dirPath.Replace(cleanPath, string.Empty).Replace(@"\", "/") : dirPath.Replace(@"\", "/");

Console.WriteLine(string.Format("Diretorio {0} replicado", currentFolder));

foreach (string filePath in Directory.GetFiles(dirPath))

{

string currentFile = Path.GetFileName(filePath);

using (Stream fileStream = File.Open(filePath, FileMode.Open))

{

pBAmazonS3.SaveObject(fileStream, string.Format(@"{0}/{1}", currentKey,   
 currentFile));

Console.WriteLine(string.Format("Arquivo {0} replicado",   
 Path.GetFileName(filePath)));

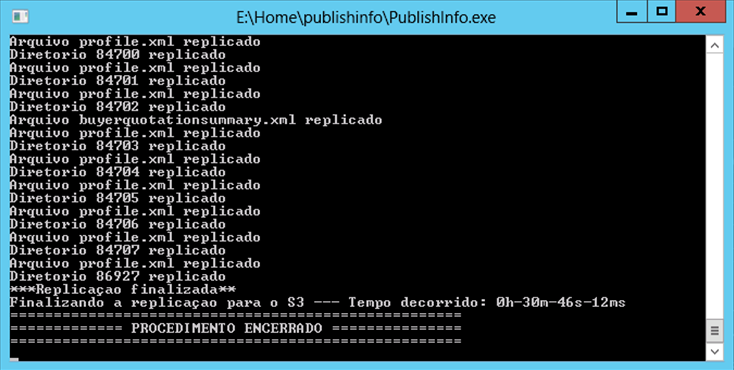
}

}

ReplicationFilesRecursive(dirPath, pBAmazonS3, cleanPath);

}

}

**Results:**  


**Using Parallell.ForEach in directories**

**Code:**

static void ReplicationFilesRecursive(string localDir, BAmazonS3 pBAmazonS3, string cleanPath = null)

{

Parallel.ForEach(Directory.GetDirectories(localDir), dirPath =>

{

string currentFolder = Path.GetFileName(dirPath);

string currentKey = cleanPath != null ? dirPath.Replace(cleanPath, string.Empty).Replace(@"\", "/") : dirPath.Replace(@"\", "/");

Console.WriteLine(string.Format("Diretorio {0} replicado", currentFolder));

foreach (string filePath in Directory.GetFiles(dirPath))

{

string currentFile = Path.GetFileName(filePath);

using (Stream fileStream = File.Open(filePath, FileMode.Open))

{

pBAmazonS3.SaveObject(fileStream, string.Format(@"{0}/{1}", currentKey,   
 currentFile));

Console.WriteLine(string.Format("Arquivo {0} replicado",   
 Path.GetFileName(filePath)));

}

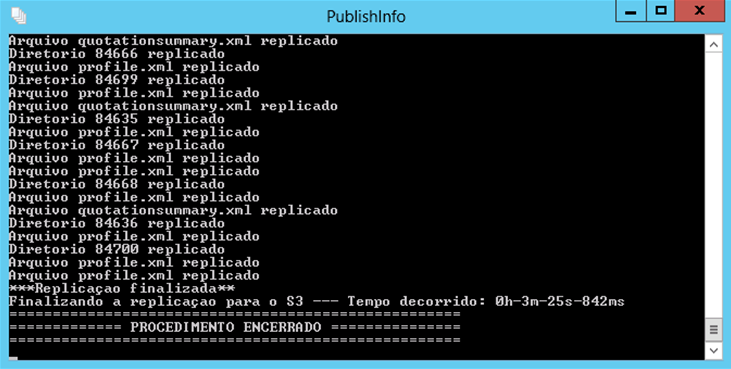
}

ReplicationFilesRecursive(dirPath, pBAmazonS3, cleanPath);

});

}

**Results:**



**Using Parallell.ForEach in directories and files**

**Code:**

static void ReplicationFilesRecursive(string localDir, BAmazonS3 pBAmazonS3, string cleanPath = null)

{

Parallel.ForEach(Directory.GetDirectories(localDir), dirPath =>

{

string currentFolder = Path.GetFileName(dirPath);

string currentKey = cleanPath != null ? dirPath.Replace(cleanPath, string.Empty).Replace(@"\",   
 "/") : dirPath.Replace(@"\", "/");

Console.WriteLine(string.Format("Diretorio {0} replicado", currentFolder));

Parallel.ForEach(Directory.GetFiles(dirPath), filePath =>

{

string currentFile = Path.GetFileName(filePath);

using (Stream fileStream = File.Open(filePath, FileMode.Open)) {

pBAmazonS3.SaveObject(fileStream, string.Format(@"{0}/{1}", currentKey,   
currentFile));

Console.WriteLine(string.Format("Arquivo {0} replicado",   
 Path.GetFileName(filePath)));

}

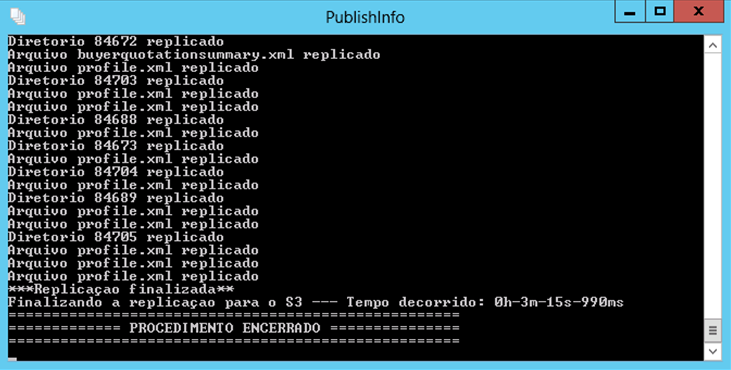
});

ReplicationFilesRecursive(dirPath, pBAmazonS3, cleanPath);

});

}

**Results:**



**Comparing the results**

|  |  |  |
| --- | --- | --- |
| Execution | Time Elapsed | Perfomance |
| Classic ForEach | 30m 46s 12ms | **-** |
| Parallell.ForEach in directories | 3m 25s 842ms | **9.00 x faster (900%)** |
| Parallell.ForEach in directories and files | 3m 15s 990ms | **9.46 x faster (946%)** |

**Conclusion**The use of *Parallel* was possible in this case because the copy of files can be executed in many threads without locked or process concurrence. Each iteration opened and release a stream file to be transfer in *BAmazonS3()* class. After that *BAmazonS3()* class just send the stream to S3 Buckets using the *SaveObjectMethod().* The gain of performance was 900% in the first case using parallelism in files iteration, and 946% in the second case using parallelism in files iteration. The gain in the second case could be higher, but in my scenario each folder had 5 files on average.