Old logic L1 pick up:

for (int idx = 0; idx < batchToProcess.Count; idx++)

{

int uid = batchToProcess[idx].UID;

string index\_path = batchToProcess[idx].Path;

string GUID = batchToProcess[idx].GUID;

// checks if index is readable and resets for reindexing if not

if (ValidateIndex(uid, index\_path, true, GUID))

{

m2iCollection.Add(batchToProcess[idx]);

doc\_count = batchToProcess[idx].SuccessCount;

new\_doc\_count += doc\_count;

//KL 04.15.2010 - Checking File Size before applying merge actions.

new\_index\_Size += DirectoryUtil.GetDirectorySize(index\_path);

}

}

1. They collect the batch and accumulate , Dco count and total sizes

Old logic L3 pickup .

public int Optimize(int groupId, MailsToIndexBatch m2iCollection, ref long new\_index\_Size, ref int new\_doc\_count, eIndexTypes indexType)

{

// the index level

int level\_num = 0;

// the list of indexes to optimize

List<CIndex> searchIndices = new List<CIndex>();

// get all the active indexes for this group and index type order by index level

using (DataSet ds = GetSearchIndex(groupId, indexType))

{

//Merge search index info into the index levels array...

if (ds != null && ds.Tables[0].Rows.Count > 0)

{

if (Logger.IsInfoEnabled)

{

Logger.InfoFormat("Found \"{0}\" active indexes for groupId=\"{1}\", index type \"{2}\" to optimize..."

, ds.Tables[0].Rows.Count

, groupId

, (int)indexType);

}

int level;

// build the list of group indexes to optimize based on index level and doc count

foreach (DataRow searchIndex in ds.Tables[0].Rows)

{

CIndex index = new CIndex();

index.index\_id = int.Parse(searchIndex["indexid"].ToString());

if (index.index\_id > 0)

{

index.index\_path = searchIndex["indexpath"].ToString();

index.locked = bool.Parse(searchIndex["IsLocked"].ToString());

// parse index level from index path (last number in path)

level = GetLevelNumber(index.index\_path);

// get the doc count if has been merged before

if (File.Exists(index.index\_path + "\_mergesuccess"))

{

int.TryParse(searchIndex["DocCount"].ToString(), out index.doc\_count);

}

// add index to list of indexes to merge (limited by max doc count)

if (!index.locked &&

index.doc\_count > 0 &&

index.doc\_count < ConfigHost.MaxDocCount &&

!string.IsNullOrEmpty(index.index\_path))

{

if (Logger.IsDebugEnabled)

{

Logger.DebugFormat("Optimizing...adding index \"{0}\" with docCount=\"{1:N0}\" to merge list...",

index.index\_path

, index.doc\_count);

}

searchIndices.Add(index);

}

else

{

Logger.DebugFormat("Excluding...index \"{0}\" with docCount=\"{1:N0}\" IsLocked={2} from merge list...",

index.index\_path

, index.doc\_count, index.locked);

}

}

}

if (Logger.IsInfoEnabled)

{

Logger.InfoFormat("Number of indexes to \"optimize\" is \"{0}\", each with docCount < maximum=\"{1:N0}\""

, searchIndices.Count

, ConfigHost.MaxDocCount);

}

int indexShardsMergingUp = 1;

// compared the group indexes in the database with the physical files and build a list of indexes to merge

// restricted by index max size and document count limits.

// Search Index list is ordered by index level

for (int i = 0; i < searchIndices.Count; i++)

{

// KL Added to prohibit large index growth.

if (indexShardsMergingUp > ConfigHost.MaxIndexShardsToMerge)

{

if (Logger.IsInfoEnabled)

{

Logger.InfoFormat("Skipping remaining indexes...reached index merge limit of \"{0:N0}\""

, ConfigHost.MaxIndexShardsToMerge);

}

break;

}

// check current indexes to merge working set document number is less than maximum (typically 400,000 docs)

else if (m2iCollection.TotalSuccessCount > ConfigHost.MaxDocCount)

{

if (Logger.IsInfoEnabled)

{

Logger.InfoFormat("Skipping remaining indexes...reached index merge doc count maximum \"{0:N0}\""

, ConfigHost.MaxDocCount);

}

break;

}

// check new index document number plus existing index doc count is less than maximum

else if (new\_doc\_count + searchIndices[i].doc\_count > ConfigHost.MaxDocCount)

{

if (Logger.IsInfoEnabled)

{

Logger.InfoFormat("Skipping index \"{0}\"...new index docCount=\"{1:N0}\" plus current index docCount=\"{2:N0}\" > maximum \"{3:N0}\""

, searchIndices[i].index\_path

, new\_doc\_count

, searchIndices[i].doc\_count

, ConfigHost.MaxDocCount);

}

continue;

}

// check existing index doc count is less than maximum

else if (searchIndices[i].doc\_count > ConfigHost.MaxDocCount)

{

if (Logger.IsInfoEnabled)

{

Logger.InfoFormat("Skipping index \"{0}\"...current index docCount=\"{1:N0}\" > maximum \"{2:N0}\""

, searchIndices[i].index\_path

, searchIndices[i].doc\_count

, ConfigHost.MaxDocCount);

}

continue;

}

if (!System.IO.Directory.Exists(searchIndices[i].index\_path))

{

Logger.WarnFormat("Skipping index...current index \"{0}\" does not exist"

, searchIndices[i].index\_path);

continue;

}

// TODO: duplicate transposed doc count check - Remove?

else if (searchIndices[i].doc\_count + new\_doc\_count > ConfigHost.MaxDocCount)

{

if (Logger.IsInfoEnabled)

{

Logger.InfoFormat("Skipping index \"{0}\"...new index docCount=\"{1:N0}\" plus current index docCount=\"{2:N0}\" > maximum \"{3:N0}\""

, searchIndices[i].index\_path

, new\_doc\_count

, searchIndices[i].doc\_count

, ConfigHost.MaxDocCount);

}

continue;

}

int indexLevelNo = GetLevelNumber(searchIndices[i].index\_path);

string indexDir = GetIndexDir(ConfigHost.WorkingFolder, groupId, indexLevelNo);

// acquire file lock on index directory

MergeLock indexLock = MergeLockFactory.AcquireLock(indexDir, batchToProcess.GUID);

if (!indexLock.IsLockAquired)

{

Logger.WarnFormat("Skipping index \"{0}\"...failed to acquire merge file lock on index"

, indexDir);

continue;

}

try

{

// check if the index is readable

if (!ValidateIndex(0, searchIndices[i].index\_path, false, string.Empty))

{

Logger.WarnFormat("Skipping index \"{0}\"...index validation failed"

, searchIndices[i].index\_path);

level\_num = GetLevelNumber(searchIndices[i].index\_path);

indexLock.Dispose();

continue;

}

long indexSize = DirectoryUtil.GetDirectorySize(searchIndices[i].index\_path); //KL 04.15.2010

level\_num = i;

// check if the index is not locked in db but is locked on file system

if (!searchIndices[i].locked && indexLock.IsLockAquired)

{

// Check the index file size + the "to be added" index file size

// and do not merge if it will exceed max file size

if (indexSize + new\_index\_Size >= ConfigHost.MaxIndexSize)

{

if (Logger.IsInfoEnabled)

{

Logger.InfoFormat("Skipping index \"{0}\"...new index size = \"{1:N0}\" KB plus current index size =\"{2:N0}\" KB > maximum \"{3:N0}\" KB"

, searchIndices[i].index\_path

, new\_index\_Size / 1024

, indexSize / 1024

, ConfigHost.MaxIndexSize / 1024);

}

indexLock.Dispose();

continue;

}

if (searchIndices[i].index\_path != "")

{

lock (((ICollection)m2iCollection).SyncRoot)

{

// add new index to mailsToIndex merge working set if it does not already exist

if (!m2iCollection.Any(x => x.UID == -searchIndices[i].index\_id))

{

if (Logger.IsInfoEnabled)

{

Logger.InfoFormat("Adding Index Id=\"{0}\" to merge working set, Path=\"{1}\", GroupId=\"{2}\", Document Count=\"{3:N0}\"..."

, searchIndices[i].index\_id

, searchIndices[i].index\_path

, groupId

, searchIndices[i].doc\_count);

}

// add index to be merged

m2iCollection.Add(new MailsToIndexItem()

{

UID = -searchIndices[i].index\_id,

Path = searchIndices[i].index\_path,

GroupID = groupId,

IsMailsToIndexRecord = false

});

indexShardsMergingUp++; // increment the number of shards merged

}

}

new\_doc\_count += searchIndices[i].doc\_count;

new\_index\_Size += indexSize; //Kl 04.15.2010

}

else

{

Logger.WarnFormat("Skipping index ID=\"{0}\"...index path is blank \"{1}\" (searchIndexDBLock={2}, fileLock={3})"

, searchIndices[i].index\_id

, searchIndices[i].index\_path

, searchIndices[i].locked

, indexLock.IsLockAquired);

indexLock.Dispose();

}

}

level\_num++; //in case there's no more levels, set a new level to create

}

catch (Exception ex)

{

Logger.ErrorFormat("Error adding index ID=\"{0}\", index path=\"{1}\" to merge working set"

, searchIndices[i].index\_id

, searchIndices[i].index\_path);

Logger.Error(ex);

throw;

}

} // end for

}

}

return level\_num;

}

Steps

1. Pick up the list of all active L3 indexes
2. If the L3 index not locked doccount>0 and doccount < 300k and indexpath not null Added to List A
3. Go through List A
4. If you reach the total Max Shards allowed return the selected shards
5. If the current L1 indexes > 300k return don’t pick any L3
6. If the L1 doccount + the L3 docount > 300k skip this index ; continue
7. If the L3 dococunt > 300k skip this index ; continue
8. If the path of the L3 not exist , skip it ; continue
9. If the L3 doccount + the L1 docount > 300k skip this index ; continue (duplicate check).
10. If Acquire Lock file on L3 index fail , skip this index , continue;
11. If the Index Size + new Index Size (accumulated) > than the MaxIndexSize 3 GB skip this index
12. Otherwise take this index , accumulate new doccount ,accumulate new index size
13. Try the above logic again

The hot fix is :

1. Calculate L1 uids Size if it greater than the MaxIndexSize 3gb don’t use L3 shards
2. Accumulate the selected L3 size with L1 Sizes till we reach 3 gb don’t take any more shards.

However over time we are growing in Mail Volum ,Indexing volum merge volum

Once we reach the point to add more indexing servers and Merge servers

3 GB file will become again a problem in the network as we grow our network and bandwidth should grow

Or have clustered servers from 1-20 on a dedicate it network from 20-40 on another dedicated network.