Problème ouvert de concaténation de grandes tables de données
(essais d'optimisation de requêtes SAS,
suivi de la conference de M. Jean Hardy en octobre 2015 au Club SAS de Québec)

Francois Ouellet <francois.ouellet@retraitequebec.gouv.qc.ca>

Mise en contexte

Différents groupes d'indicateurs sont produits en sortie dans des tables SAS multiples. Pour chaque groupe d'indicateurs, nous avons 20 tables SAS, chacune de ces tables contient un intervalle précis de numéro de clients. Exemple :

Table groupe indicateurs A clients no 00000001 à 50000000
Table groupe indicateurs A clients no 50000001 à 10000000
Etc. jusqu'à 20 tables pour groupe indicateurs A

Table groupe indicateurs B clients no 00000001 à 50000000
Table groupe indicateurs B clients no 50000001 à 10000000
Etc. jusqu'à 20 tables pour groupe indicateurs B

Une fois que toutes les tables sont produites, on joint ensemble les données :

- MERGE des tables des différents groupes d'indicateurs par numéro de client ce qui donne 20 tables qu'on appelle partitions :
  - Partition 1 : tous les indicateurs pour clients no 00000001 à 50000000
  - Partition 2 : tous les indicateurs pour clients no 50000001 à 10000000
  - Etc. jusqu'à 20 partitions
- SET de toutes les partitions pour obtenir une table unique en sortie qui contient environ 300 millions de rangées et 80 colonnes
- Création d'un INDEX par numéro de client sur la table de sortie, requis pour interrogation des données par l'utilisateur

La seconde étape (le SET) peut prendre plus de 40 minutes à réaliser. Lors de la rencontre du club SAS en octobre dernier, des participants ont fourni des pistes d'optimisation possibles. Nous en avons évalué plusieurs. Celle qui consistait à faire moins de partitions a porté fruit. Voici les explications.

Pistes d'optimisation

1) Faire moins de partitions

Lorsque le traitement a été développé à l'époque, le système de partitions a été mis en place dans le but de faire du parallélisme, car plusieurs partitions pouvaient être prises en charge par des processus différents en même temps. Également, c'était possible de reprendre seulement les partitions manquantes en cas d'interruption du traitement. Nous avons modifié cette approche afin de retirer le concept des partitions et que le parallélisme porte plutôt plusieurs groupes d'indicateurs produits simultanément. Lorsqu'un groupe d'indicateurs est produit, nous le faisons pour tous les numéros de clients à la fois. Ce qui donne :

Table groupe indicateurs A contenant tous les clients
Table groupe indicateurs B contenant tous les clients
Etc.

Notre enjeu de performance sur un SET est devenu un enjeu sur un MERGE des tables d'indicateurs qui prend environ le même temps, soit 40 minutes, alors que les étapes préalables ont été optimisées par l'élimination des partitions.

À noter également que:
- des tests ont été faits pour trouver un bon compromis entre temps de production des fichiers et espace disque utilisé, pour chaque table, via les options de compression (compress = no | char | binary)
- plutôt que d'effectuer le merge, nous avons tenté de conserver les tables d'indicateurs séparément et de créer un index sur chacune. L'interrogation des données se serait ensuite effectuée avec un PROC SQL simple qui joint toutes les tables ensemble tout en utilisant un filtre sur le numéro de client. Cependant le 40 minutes épargné au niveau du merge a été remplacé par un temps équivalent au niveau de la création des index et la taille totale des fichiers était supérieure, c'est pourquoi cette solution n'a pas été retenue.

2) Data step views

Les vues permettent de fusionner ou concaténer les données sans produire un nouveau fichier physique. Elles sont créées instantanément. Exemple :

```
data W.test_data_view / view=W.test_data_view;
  merge W.s_2em
        W.s_1er
        W.s_pia
        W.s_ria
        W.s_qgi;
  by no_seq_cli annee;
run;
```

Par contre l'interrogation subséquente de la vue pour un numéro de client demande trop de temps, puisque la vue réfère à des tables contenant 300 millions d'occurrences.

```
data test;
   set W.test_data_view;
    where no_seq_cli=3500010;
run;
```

3) Options Bufsize et Bufno

La modification des paramètres bufsize et bufno n'a pas amélioré le temps de traitement. Différentes valeurs de paramètres ont été utilisées. Voici celles suggérées par SAS :

http://support.sas.com/kb/46/954.html

4) Hash tables

Notre compréhension des tables de hash est qu'elles sont surtout utiles pour effectuer des mises à jour à partir d'un dataset transactionnel de petite taille vers une table maître. Puisque les objets hash sont créés en mémoire, on ne peut les utiliser pour des tables de volumes comparables aux nôtres sans obtenir des erreurs de mémoire insuffisante. Voici un exemple d'utilisation :

```
data W.match;
        /*length k no_seq_cli;
        length s 8;*/
        if _N_ = 1 then do;
          /* load SMALL data set into the hash object */
         declare hash h(dataset: "W.s_sim2016_ace_amg_042016_2em");
          /* define SMALL data set variable K as key and S as value */
          h.defineKey('no_seq_cli','annee');
          h.defineDone();
          /* avoid uninitialized variable notes */
          /*call missing(k, s);*/
        end;

      /* use the SET statement to iterate over the LARGE data set using */
      /* keys in the LARGE data set to match keys in the hash object */
      set W.s_sim2016_ace_amg_042016_pia;
      rc = h.find();
      if (rc = 0) then output;
run;
```

Également, au même titre que les tables de hash, la fonction SAS modify peut être utile si on doit mettre à jour une table volumineuse à partir d'une table transactionnelle plus petite.

Conclusion

L'élimination des partitions a permis d'optimiser le traitement global. Parmi les autres pistes que nous allons évaluer dans le futur, il y a le chargement en SQL Server des données en « bulkload » via ODBC, ainsi que l'amélioration du lien réseau par lequel transitent les données lors du merge, car les opérations d'écriture sont moins rapides que sur le disque local du serveur SAS.