Documentation Outil de pack IPSL

1- Objectif et contexte

2- Rappel sur les différents types de fichiers à traiter

3- La création des listes de fichiers à traiter

4- Le traitement des données

5- Mode d’emploi

1) Objectif :

Transférer sur le cccstoredir et le cccworkdir les simulations stockées sur le dmnfs. Nous voulons que le format de stockage à l'arrivée soit conforme au nouveau format mis en place pour la production sur ces disques. Pour cela, l’IPSL a développé un outil de traitement des données dmnfs. Cet étape se réalise en 2 sous-étapes : la création des listes de fichiers à traiter à une certaine fréquence d’une part et le traitement de ces listes de fichiers d’autre part.

Attention, cet outil fonctionne uniquement lorsque les répertoires des données d’entrée et les répertoires de données de sortie sont situés sur le même compte utilisateur.

2) Rappel des différents types de fichiers créés par une simulation :

- Output au format netcdf
  - Ancien format : copiés mensuellement dans les répertoires JobName/COMP/Output/*/
  - Nouveau format : concaténés (ncrcat) par période dans les répertoires JobName/COMP/Output/*/
- Restart au format netcdf
  - Ancien format : copiés mensuellement dans les répertoires JobName/COMP/Restart/
  - Nouveau format : archivés (tar) par période dans les répertoires JobName/RESTART/ (attention on renomme les fichiers)
- Debug au format texte
  - Ancien format : copiés mensuellement dans les répertoires JobName/COMP/Debug/
  - Nouveau format : archivés (tar) par période dans les répertoires JobName/DEBUG/ (attention on renomme les fichiers)

- Post-traitements : il en existe deux types, les Analyses qui seront copiées telles quelles sur le $CCCSTOREDIR et les Monitoring et les Atlas qui seront eux copiés sur le $CCCWORKDIR.

3) La création des listes de fichiers à traiter

La création des listes de fichiers à traiter est réalisée par le script « launch_ipsl_pack.sh ». Ce script parcourt l'espace des données initiales et les découpe sous la forme d'un ensemble de listes de fichiers. Une liste de fichiers est un fichier au format texte. Chaque ligne de ce fichier est constituée du chemin absolu d'un fichier de données.

Le script « launch_ipsl_pack.sh » établit les listes de fichiers de données à traiter et les range dans une arborescence similaire à celles des données initiales, sous le répertoire ${IGCM_DEM}.

Description

Dans cette documentation, la variable JobName représentera le nom d'une simulation. Ci-après une description des différents scripts qui composent l'outil de création des listes.

a. DEM_utilities.sh

Ensemble de fonctions utilisées dans les scripts du Pack IPSL, permettant entre autre de gérer la création de log.

b. launch_ipsl_pack.sh

C'est le script principal de lancement de l'outil permettant la création des listes des fichiers à packer.

Voici la liste des scripts dans leur ordre d'appel :

- create_listing.sh
- find_directory_simul.sh
- create_config_card.sh
- calcul_size_simul.sh
- find_size_pack.sh
- write_liste_pack.sh
- archive_restart.sh
- archive_debug.sh

Ce script prend en argument d'entrée un fichier texte contenant le(s) path(s) d'un répertoire de type IGCM_OUT/ ou d'un sous-répertoire contenant une à plusieurs simulations.

Exemple de lancement :

```
>> ./launch_ipsl_pack.sh param_AC.txt

avec

>> vi param_AC.txt

>>> /dmnfs11/cont003/p86cozic/IGCM_OUT
```

Dans les scripts ce fichier (dans notre exemple param_AC.txt) est nommé ${LISTE_SIMUL}

L'ensemble des scripts crée un répertoire IGCM_DEM/IGCM_OUT/... dans lequel nous stockerons les listes créées pour chaque simulation étudiée. Et un répertoire IGCM_DEM/tmp/ dans lequel nous travaillons.

c. create_listing.sh

Ce script prend en argument d'entrée ${LISTE_SIMUL}

Pour pouvoir travailler en interrogeant un minimum l'archive nous avons décidé de créer dès le début de l'exécution un listing des fichiers existants sous le(s) path(s) pour lesquels nous voulons appliquer le Pack. Nous exploiterons constamment par la suite ce listing. Il comprend trois colonnes

- 1ère colonne : type de fichier (directory – file – link)
- 2ième colonne : taille du fichier
- 3ième colonne : path complet du fichier

Le listing créé est $SCRATCHDIR/IGCM_DEM/Listing.txt. Dans les scripts il est définit pas la variable ${LISTE_DMNFS}.

Dans notre exemple les premières lignes en sont :

```
d 130 /dmnfs11/cont003/p86cozic/IGCM_OUT

d 129 /dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1

d 86 /dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1/ESM_06

d 45 /dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1/ESM_06/ATM

d 15 /dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1/ESM_06/ATM/Output

d 4096 /dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1/ESM_06/ATM/Output/MO
```

```
f                                                                                                                                                                                        19419800
/dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1/ESM_06/ATM/Output/MO/ESM_06_19900101_1990
0130_1M_histmth.nc

f                                                                                                                                                                                        19419800
/dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1/ESM_06/ATM/Output/MO/ESM_06_19900201_1990
0230_1M_histmth.nc

f                                                                                                                                                                                        19419800
/dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1/ESM_06/ATM/Output/MO/ESM_06_19900301_1990
0330_1M_histmth.nc

f                                                                                                                                                                                        19419800
/dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1/ESM_06/ATM/Output/MO/ESM_06_19900401_1990
0430_1M_histmth.nc

f                                                                                                                                                                                        19419800
/dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1/ESM_06/ATM/Output/MO/ESM_06_19900501_1990
0530_1M_histmth.nc

f                                                                                                                                                                                        19419800
/dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1/ESM_06/ATM/Output/MO/ESM_06_19900601_1990
0630_1M_histmth.nc

f                                                                                                                                                                                        19419800
/dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1/ESM_06/ATM/Output/MO/ESM_06_19900701_1990
0730_1M_histmth.nc

f                                                                                                                                                                                        19419800
/dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1/ESM_06/ATM/Output/MO/ESM_06_19900801_1990
0830_1M_histmth.nc

f                                                                                                                                                                                        19419800
/dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1/ESM_06/ATM/Output/MO/ESM_06_19900901_1990
0930_1M_histmth.nc
```

Traitement des cas particuliers :

- Nous retirons directement du listing les fichiers contenus dans des répertoires SPIN/ qui seront traités ultérieurement par le CCRT.
- Nous retirons les fichiers qui sont des liens (repérés par un « l » en première colonne)
- Nous retirons les fichiers de type run.card qui seront traités ultérieurement par le CCRT.

d. find_directory_simul.sh

Ce script prend en argument d'entrée le fichier ${LISTE_SIMUL} et ${LISTE_DMNFS}.

Il permet pour chacun des paths inscrits dans ${LISTE_SIMUL} de lister l'ensemble des simulations situées en dessous dans l'architecture. Nous considérons que nous avons une simulation si il y a au moins un répertoire Restart/ dedans.

Cette nouvelle liste est disponible dans le fichier $SCRATCHDIR/IGCM_DEM/liste_simul_${LISTE_SIMUL_NAME}

avec

LISTE_SIMUL_NAME=$(basename ${LISTE_SIMUL})

Dans notre exemple : $SCRATCHDIR/IGCM_DEM/liste_simul_param_AC.txt Et les premières lignes de ce fichier sont : /dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1/ESM_06 /dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1/ESM_07 /dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1/ESM_08 /dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1/ESM_102 /dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1/ESM_Atl /dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1/ESM_200 /dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1/ESM_201 /dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1/CM4_200 /dmnfs11/cont003/p86cozic/IGCM_OUT/IPSL_ESM_V1/CM4_201 /dmnfs11/cont003/p86cozic/IGCM_OUT/LMDZINCA/AER/AER_REF /dmnfs11/cont003/p86cozic/IGCM_OUT/LMDZINCA/AER/AER_REF2 /dmnfs11/cont003/p86cozic/IGCM_OUT/LMDZINCA/AER/AER_MERGE /dmnfs11/cont003/p86cozic/IGCM_OUT/LMDZINCA/AER/AER_v2

e. create_config_card.sh

Ce script prend en arguments d'entrée

$SCRATCHDIR/IGCM_DEM/liste_simul_${LISTE_SIMUL_NAME} et ${LISTE_DMNFS}

Ce script permet pour chacune des simulations listées de recréer sa carte d'identité « config.card » (${IGCM_DEM_SIMU}/config_card_${JobName}). Cette carte contiendra les informations suivantes :

- JobName : nom de la simulation
- DateBegin : date de début de la simulation
- DateEnd : date de fin de la simulation
- PATH_SIMU_FULL : path de la simulation sur le dmnfs
- IGCM_DEM_SIMU : path sur le scratchdir dans lequel nous stockerons les listes.

Pour trouver DateBegin et DateEnd nous utilisons les noms des fichiers Restart et Output créés par une simulation. Chacun de ces fichiers contient dans son nom la date du mois de simulation auquel il correspond. On conserve en DateBegin la plus petite date trouvée et en DateEnd la plus grande date trouvée lorsque l'on passe en revue tous ces fichiers.

Si nous considérons la simulation

/dmnfs11/cont003/p86cozic/IGCM_OUT/LMDZORINCA/AER/AER_AR5WERF_VALID

Sa carte sera :

$SCRATCHDIR/IGCM_DEM/IGCM_OUT/LMDZORINCA/AER/AER_AR5WERF_VALID/config_card_AER_AR5WERF_VALID

et elle contiendra les informations suivantes :

JobName=AER_AR5WERF_VALID

DateBegin=20000101

DateEnd=20001231

PATH_SIMUL_FULL=/dmnfs11/cont003/p86cozic/IGCM_OUT/LMDZORINCA/AER/AER_AR5WERF_VALID

IGCM_DEM_SIMU=/scratch/cont003/p86cozic/IGCM_DEM/IGCM_OUT/LMDZORINCA/AER/AER_AR5WERF_VALID

L'ensemble des cartes créées sont listées dans le fichier $SCRATCHDIR/IGCM_DEM/config_card.liste. Il contient deux colonnes :

- 1ère colonne : pour chaque simulation contenue dans

  liste_simul_${LISTE_SIMUL_NAME} le path de sa carte config.card

- 2ième colonne : un log permettant le management des scripts. A ce stade «*ListToBeDone* »

Dans notre exemple les premières lignes de ce fichier sont

/scratch/cont003/p86cozic/IGCM_DEM/IGCM_OUT/IPSL_ESM_V1/ESM_06/config_card_ESM_06 ListToBeDone

/scratch/cont003/p86cozic/IGCM_DEM/IGCM_OUT/IPSL_ESM_V1/ESM_07/config_card_ESM_07 ListToBeDone

/scratch/cont003/p86cozic/IGCM_DEM/IGCM_OUT/IPSL_ESM_V1/ESM_08/config_card_ESM_08 ListToBeDone

/scratch/cont003/p86cozic/IGCM_DEM/IGCM_OUT/IPSL_ESM_V1/ESM_102/config_card_ESM_102 ListToBeDone

/scratch/cont003/p86cozic/IGCM_DEM/IGCM_OUT/IPSL_ESM_V1/ESM_Atl/config_card_ESM_Atl ListToBeDone

/

Le script create_config_card permet également de créer, pour chaque simulation, les listings par type de fichiers et commence à traiter certains cas particuliers.

Les listing créés et qui seront utilisés par les scripts suivants sont :

- La liste des fichiers de restart (on conserve uniquement les colonnes 2 et 3 du Listing)

  ${IGCM_DEM_SIMU}/liste_restart_files_config.txt

Si l'on reprend notre exemple les premières lignes de cette liste sont du type :

49905924 /dmnfs11/cont003/p86cozic/IGCM_OUT/LMDZORINCA/AER/AER_AR5WERF_VALID/ATM/Restart/AER_AR5WERF_VALID_20000131_restart.nc

16040520 /dmnfs11/cont003/p86cozic/IGCM_OUT/LMDZORINCA/AER/AER_AR5WERF_VALID/ATM/Restart/AER_AR5WERF_VALID_20000131_restartphy.nc

49905924 /dmnfs11/cont003/p86cozic/IGCM_OUT/LMDZORINCA/AER/AER_AR5WERF_VALID/ATM/Restart/AER_AR5WERF_VALID_20000229_restart.nc

16040520
/dmnfs11/cont003/p86cozic/IGCM_OUT/LMDZORINCA/AER/AER_AR5WERF_VALID/ATM/Restart/AER_AR5WERF_VALID_20000229_restartphy.nc

- La liste des outputs (on conserve uniquement les colonnes 2 et 3 du Listing) :

  ${IGCM_DEM_SIMU}/liste_output_files_config.txt

Si l'on reprend notre exemple les premières lignes de cette liste sont du type

1538315256
/dmnfs11/cont003/p86cozic/IGCM_OUT/LMDZORINCA/AER/AER_AR5WERF_VALID/ATM/Output/MO/AER_AR5WERF_VALID_20000101_20000131_1M_histmth.nc

1439077304
/dmnfs11/cont003/p86cozic/IGCM_OUT/LMDZORINCA/AER/AER_AR5WERF_VALID/ATM/Output/MO/AER_AR5WERF_VALID_20000201_20000229_1M_histmth.nc

1538315256
/dmnfs11/cont003/p86cozic/IGCM_OUT/LMDZORINCA/AER/AER_AR5WERF_VALID/ATM/Output/MO/AER_AR5WERF_VALID_20000301_20000331_1M_histmth.nc

1488696280
/dmnfs11/cont003/p86cozic/IGCM_OUT/LMDZORINCA/AER/AER_AR5WERF_VALID/ATM/Output/MO/AER_AR5WERF_VALID_20000401_20000430_1M_histmth.nc

1538315256
/dmnfs11/cont003/p86cozic/IGCM_OUT/LMDZORINCA/AER/AER_AR5WERF_VALID/ATM/Output/MO/AER_AR5WERF_VALID_20000501_20000531_1M_histmth.nc

1488696280
/dmnfs11/cont003/p86cozic/IGCM_OUT/LMDZORINCA/AER/AER_AR5WERF_VALID/ATM/Output/MO/AER_AR5WERF_VALID_20000601_20000630_1M_histmth.nc

De ces deux listes on retire tous les fichiers qui ne sont pas du type $JobName_$date ou qui sont dans des sous-répertoires de Restart/ ou de Output/*/. Ces éventuels fichiers ne sont pas créés par libIGCM et nous ne savons pas comment les traiter. Leurs listes sont actuellement stockées dans les fichiers ${IGCM_DEM_SIMU}/other_tar/tar_no_output_files.list et

${IGCM_DEM_SIMU}/other_tar/tar_no_restart_files.list . Nous ne savons pas encore ce que nous ferons de ces listes.

Nous créons également deux répertoires ${IGCM_DEM_SIMU}/work_cp/ et

${IGCM_DEM_SIMU}/store_cp/ dans lesquels nous stockerons les listes de fichiers à copier sans rien modifier entre le dmnfs et le nouveau système de stockage. Ces listes contiennent les Monitoring, les Atlas, les fichiers d'Analyse et le répertoire Exe/. Ainsi que pour certaines simulations un fichier nommé mesh_mask.nc qui est unique et ne peut donc être packé avec rien d'autre.

Si la simulation ne contient pas de fichiers de type Output nous considérons qu'elle n'est pas traitable et nous l'archivons entièrement via tar (appelle à write_liste_tar.sh, voir documentation plus bas).

Désormais nous allons travailler sur chacune des cartes de config créées (${CONFIG})

f. calcul_size_simul.sh

Ce script prend en argument une carte de config ${CONFIG} et le listing ${LISTE_DMNFS}.

Si une simulation a une taille de stockage inférieure à 1Go nous décidons de l'archiver via tar sans utiliser la concaténation des sorties.

Pour cela nous appelons le script write_liste_tar.sh

g. write_liste_tar.sh

Ce script prend en argument d'entrée une carte de type config.card. Il crée un fichier

${IGCM_DEM_SIMU}/tar_full_simul.txt qui contient le path de la simulation à tarer. Il indique l'état *WriteListTarDone* dans le fichier $SCRATCHDIR/IGCM_DEM/config_card.liste.

h. find_size_pack.sh

Ce script prend en argument d'entrée une carte de config ${CONFIG} et le listing

${LISTE_DMNFS}.

Dans ce script nous travaillons avec la liste de fichiers d'Output créé précédemment :

${IGCM_DEM_SIMU}/liste_output_files_config.txt

Il permet de déterminer pour une simulation donnée la fréquence de pack optimal. Pour trouver cette fréquence nous donnons une contrainte : qu'elle soit la même pour tous les types de fichiers.

Nous commençons par définir une taille de pack idéale : entre 20 et 70 Go, et une fréquence idéale : 20 ans. Ensuite pour chaque type de fichier d'output nous vérifions si pour la fréquence idéale nous respectons la taille idéale.

Si pour la fréquence idéale la taille obtenue pour un pack est trop grande ou trop petite nous appliquons une règle de trois pour obtenir la nouvelle fréquence répondant au critère de taille idéale. Au final nous retenons la plus petite fréquence calculée quelque soit le type de fichier d'Output.

Quelques cas particuliers sont traités :

- nous n'utilisons pas la taille calculée pour la dernière période (car si la simulation a un nombre d'années qui n'est pas un multiple de 20 alors la taille de la dernière période est obligatoirement plus petite que les précédentes)
- si nous n'avons qu'une seule période (donc pour les simulations dont le nombre d'années est inférieur à 20) nous recalculons la taille de la période avant de recalculer la fréquence idéale
- Si la fréquence idéale est inférieure à 1 an nous arrondissons à 1.
- Nous ramenons quoiqu'il arrive la fréquence idéale à l'une des suivantes : 1 – 5 – 10 – 20 – 50 – 100

Au final la fréquence qui vient d'être calculée est stockée dans le fichier

${IGCM_DEM_SIMU}/period_pack.txt.

i. write_list_pack.sh

Comme les scripts précédents celui ci prend en arguments d'entrée un fichier de config ${CONFIG} et le listing ${LISTE_DMNFS}.

Ce script écrit les listes de fichiers d'output à concaténer, en respectant la fréquence calculée par find_size_pack.sh. Si dans une période il manque un fichier nous archivons cette période pour ce type de fichier au lieu de la concaténer. Nous créons deux répertoires

${IGCM_DEM_SIMU}/output_ncrcat/ qui contiendra les listes à concaténer avec ncrcat, et

${IGCM_DEM_SIMU}/output_tar/ qui contiendra les listes à archiver avec tar.

Les listes créées sont nommées :

${JobName}_${datebeginperiod}_${dateendperiod}_${typefichier}.list avec

- JobName : le nom de la simulation
- datebeginperiod : la date de début de la période packée
- dateendperiod : la date de fin de la période packée
- typefichier : le type de fichier d'ouput traité dans cette liste (le nom contient l'extension .nc)

j. archive_restart.sh

Ce script prend en arguments d'entrée une carte de config ${CONFIG} et le listing

${LISTE_DMNFS}.

Ce script va permettre de créer la liste des restarts à archiver. Il utilise la liste

${IGCM_DEM_SIMU}/liste_restart_files_config.txt créée par create_config_card.sh et la fréquence de pack calculée par find_size_pack.sh.

En plus d'être regroupé par période les fichiers de restart sont renommés et stockés dans un nouveau répertoire JobName/RESTART/.

Exemple pour les fichiers renommés

Ancien stockage :

JobName/ATM/Restart/JobName_ouput1.nc

JobName/SRF/Restart/JobName_output2.nc

Nouveau stockage :

JobName/RESTART/JobName_restart_debutperiode_finperiode.tar

avec

>> tar tf JobName_restart_debutperiode_finperiode.tar

>>ATM_JobName_output1.nc

>>SRF_JobName_output2.nc

Au final les listes de restart à archiver sont dans le répertoire ${IGCM_DEM_SIMU}/restart_tar/.

k. archive_debug.sh

Ce script est calqué sur le précédent mais il traite les fichiers stockés anciennement dans les répertoires Debug/ et Out/. Au final les listes créées sont stockées dans le répertoire

${IGCM_DEM_SIMU}/debug_tar/.

l. Résumé de l'état final après passage de l'outil

Dans le répertoire ${IGCM_DEM_SIMU} nous avons créé les directories suivant :

work_cp

>> ls work_cp

>> cp_files.list

>> vi cp_files.list

>> liste des répertoires et des fichiers à copier directement sur le cccworkdir

store_cp

>> ls store_cp

>> cp_files.list

>> vi cp_files.list

>> liste des répertoires et des fichiers à copier directement sur le cccstoredir.

restart_tar

>> ls restart_tar

>> listes, à une fréquence donnée, des fichiers de restart à archiver

debug_tar

>> ls debug_tar

>> listes, à une fréquence donnée, des fichiers de debug à archiver

output_ncrcat

>> ls output_ncrcat

>> listes, à une fréquence donnée, des fichiers d'output à concaténer

output_tar

>> ls output_tar

>> listes, à une fréquence donnée, des fichiers d'ouputs à archiver

1- parce qu'il manque un fichier dans la période.

2- parce qu'il y a eu une erreur lors de la concaténation

RESTART/

>> ls RESTART

>> liens vers les fichiers Restart de la simulation en les renommant

DEBUG/

>> ls DEBUG/

>> liens vers les fichiers Debug de la simulation en les renommant

other_tar/

>> ls other_tar

>> contient des listes de fichiers qui ne sont pas produits par libIGCM et qui seront laissés dans l'espace de démigration avant d'être traités par le CCRT.

tar_full_simul.txt

Ce fichier n'existe que si la simulation est inférieure à 1Go ou si elle ne contient pas de fichiers d'Output (simulation plantée).

4) Le traitement des données

Objectif

A partir des listes de fichiers créées, on veut réaliser les étapes correspondantes :

- concaténation à base de la commande « nco » : ncrcat
- archivage avec la commande tar
- copie de fichiers directement vers le CCCSTOREDIR ou CCCWORKDIR.

Fonctionnement

Il existe deux versions de l'outil de traitement des données :

- une version parallèle

- une version séquentielle, composée d'un script principal « launch_ipsl_enlarge.sh » qui lance un script secondaire « enlarge_my_files ». Le fonctionnement de la version séquentielle est similaire au fonctionnement de la version parallèle et ne sera pas détaillé ici.

L'outil « parallèle » réalise les traitements élémentaires de pack (concaténations, opérations de tar, copie) en parallèle. Il est composé d'un script principal « parallelPack.sh » qui lance un script secondaire « launch_and_mesure_time.sh » qui va gérer et lancer les différentes tâches élémentaires via l'outil parallèle « glost ».

Le script « process_list.sh » est le script qui va réaliser les commandes élémentaires de traitement sur les fichiers. On boucle sur les directories obtenus en fin d'étape de création de listes et pour chacun d'entre eux on traite les différentes listes de fichiers.

Un fichier de log et un fichier status sont crées pour chacune des listes à traiter dans le répertoire dans lequel se trouvent les listes. Les états possibles sont : COMPLETED, FAILED ou DELEGATE (dans le cas ncrcat seulement).

Architecture des scripts de traitements des données

Ce qui est dit ici est résumé sur la **Erreur ! Source du renvoi introuvable.**.

Les caractéristiques principales de cette architecture sont les suivantes :

- les opérations de traitements des données se font dans « process_lists.sh »
- l'outil « glost » est celui qui permet de lancer le traitement des listes de fichiers de données en simultané
- le « launcher » est un script qui s'auto-resoumet dés que son temps d'exécution atteint une limite fixé par l'utilisateur des scripts (voir les fichiers configurations). Le temps d'exécution en batch d'un script ne pouvant excéder 24h, ce dispositif évite l'arrêt brutal des traitements et permet de les poursuivre « proprement ».

Le script de traitements des données est « parallelPack.sh ». C'est un script pilote qui, entre autre :

- prend en compte les variables des fichiers de configurations
- vérifie le chargement des modules requis pour les traitements à venir
- appelle le script « launch_and_measureTime.sh »

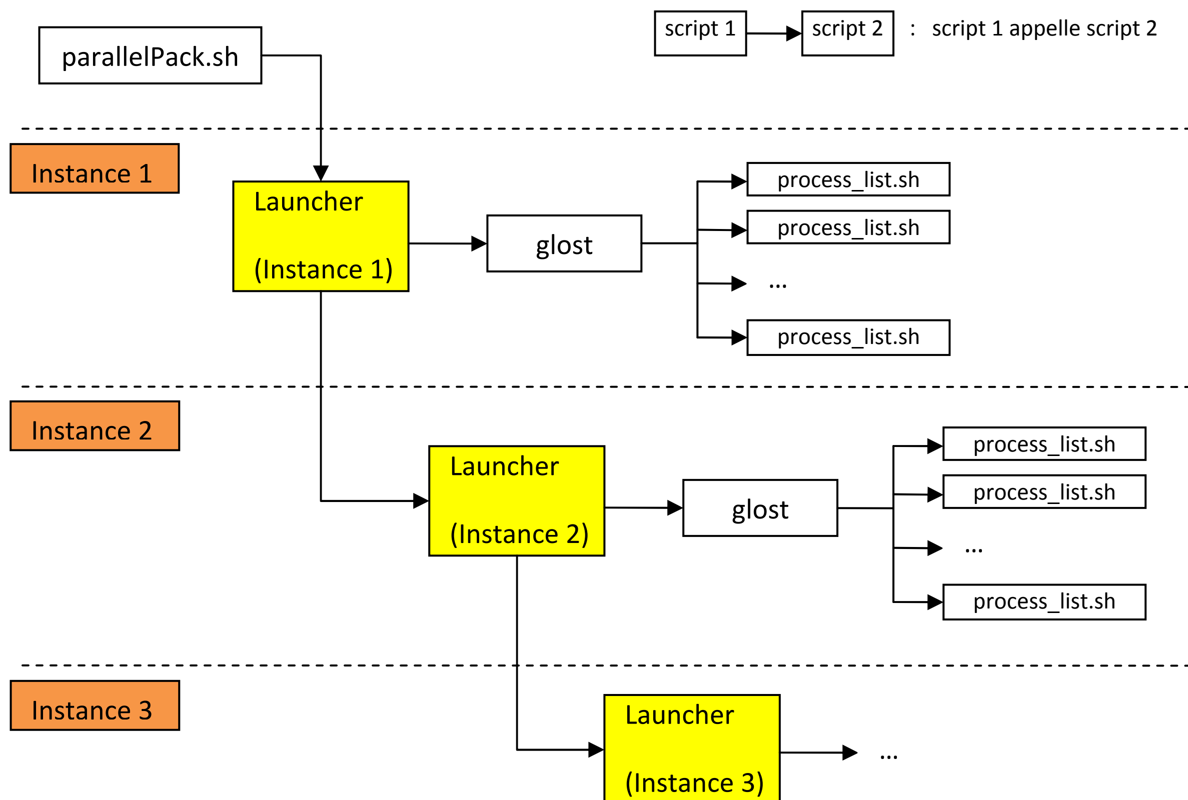

Figure 1 : schéma des appels des scripts de traitements des données d'un utilisateur

Le « launcher » (script « launch_and_measureTime.sh ») possède une caractéristique importante : il s'auto re-soumet après un temps d'exécution fixé par l'utilisateur des scripts. On appelle « instance » une exécution du « launcher ». Sur la **Erreur ! Source du renvoi introuvable.**, le script « parallelPack.sh » lance la première instance du « launcher », qui s'auto-resoumet. A ce point, l'instance 1 lance une instance 2.
Le script « launch_and_measureTime.sh » :

- reprend les traitements où ils se sont arrêtés si d'autres instances l'ont précédé ou s'il s'agit d'une nouvelle tentative de traitement sur les mêmes données (relance de « parallelPack.sh »).
- lance l'outil de traitements simultané des données
- effectue des vérifications aléatoires sur quelques unes des listes de fichiers de données traitées
- supprime les données originales qui ont été correctement traitées
- s'auto-resoumet

Il est prévu que ce script s'arrête dans 3 cas :

- arrêt brutal non prévu : disfonctionnement des machines, maintenance, …
- il n'y a plus de progression dans le traitement des données (il reste des données dont le traitement échoue systématiquement)
- toutes les données ont été traitées correctement.

« glost » est l'outil de traitement des données en « parallèle ». Il prend en argument un ensemble de commandes que lui a préparé le « launcher ».

- c'est un programme écrit en langage C utilisant MPI. Il a été compilé par le lancement de « make » préalablement.
- il lance simultanément plusieurs exécutions du script « process_list.sh » (script chargé du traitement des données proprement dit).
- au terme d'un temps fixé par l'utilisateur des scripts de traitement, il ne lance plus de nouvelles commandes et attend que les tâches en cours se terminent pour rendre la main au « launcher »

« process_list.sh » est le script qui traite les données de manière élémentaire. Il prend en argument une liste de fichiers à traiter (« packer »).

L'arborescence de suivi

L'exécution du script de traitement des données produit une arborescence qui permet de suivre le déroulement des opérations. Par exemple :

- on peut avoir relancé « parallelPack.sh »à la suite d'un arrêt non prévu, il s'agit alors d'un nouvel essai
- après un lancement, plusieurs instances du « launcher » se sont succédées

Dans ces situations, les sorties doivent être organisées et rendre compte du déroulement des évènements.
Ces sorties sont produites dans le répertoire contenu dans la variable « OUTPUT_PROGRESS » du fichier de configuration « load_ipslPack_env.sh ». Ce répertoire contient les sous répertoires correspondant à l'ensemble des simulations traitées pour un utilisateur. Sous chacun de ces sous-répertoires on trouvera un sous répertoire « TRY__${num} » par lancement du script

« parallelPack.sh » (num est le numéro du lancement ou essai). Dans chacun d'eux, on trouvera un certain nombre de fichiers dont le nom porte la trace du numéro de l'essai et du numéro de l'instance du « launcher » qui l'a produit :

- des fichiers de type « inputCmd__* » : contient les commandes que soumet l'instance courante à l'outil « parallèle » (une commande par liste de fichiers de données à traiter)
- des fichiers de type « packOutput__* » : contient notamment les codes de retour du traitement des listes proposées dans le fichier précédent.
- des fichiers de type « nextInputCmd__* » : contient les commandes que devra soumettre à l'outil de traitement « parallèle » l'instance suivante du « launcher » : la liste de commande de l'instance courante débarrassée des commandes relatives aux listes dont le traitement s'est effectué correctement
- des fichiers de type « removedFiles__* » : contient l'ensemble des données d'origine supprimées après exécution de l'instance courante.

Les différents cas de listes

output_ncrcat : listes, à une fréquence donnée, des fichiers d'output à concaténer. « ncrcat » est la commande qui permet de concaténer une série de fichiers. Plus d'infos : http://nco.sourceforge.net/nco.html#ncrcat-netCDF-Record-Concatenator

On utilisera en particulier les options suivantes:

- --md5_digest , avec la version 4.10, ncrcat effectue une vérification sur la signature des variables pour chaque record, en entrée et en sortie. La commande lève une erreur s'il y a une différence.

- -x -v aaaa,bbbb,cccc Certaines variables ne se retrouvent pas dans tous les fichiers à concaténer. Il faut donc les exclure à travers une liste à construire dynamiquement via le script bash suivant

```
nbfile=0
for file in `cat $1` ; do
   ncdump -h ${file} | gawk '{if (match($0, /(byte|char|short|int|float|double) (.*)\(/, arr)) print arr[2] }' >> tmp_$$.txt
   let nbfile=nbfile+1
done
varstoexclude=`cat tmp_$$.txt | sort | uniq -c | awk -v nbfile=$nbfile '{if ($1 != nbfile) {print $2}}' | paste -s -d','`
rm -f tmp_$$.txt
```

La commande finale lancée est :

```
ncrcat --md5_digest -x -v aaaa,bbbb,cccc
```

Si la commande renvoie un code d'erreur, le statut retourné dans le fichier status est FAILED. Par défaut, 3 tentatives sont nécessaires avant de passer à l'état DELEGATE : l'archivage se fera alors via

la commande tar plutôt que ncrcat et la liste de fichiers à traiter est déplacée dans le répertoire output_tar.

Si la commande ne renvoie pas de code d'erreur, le statut retourné dans le fichier status sera COMPLETED.

output_tar : listes, à une fréquence donnée, des fichiers d'output à archiver avec la commande tar en raison d'une erreur à la concaténation (via ncrcat).

On utilisera en particulier les options suivantes de la commande tar :

- -W (verify) : lorsque cette option est spécifiée, l'intégralité du tar est vérifiée et comparée au contenu des fichiers sources (octet par octet). Cette option est assez coûteuse puis qu'elle parcourt et lit deux fois les fichiers sources (une fois pour faire le tar, une autre fois pour comparer le contenu).
- --format=posix : qui génnère les tar au format POSIX.1-2001 Ce format supprime les limitations des formats précédents, notamment : longueur des noms de fichiers, taille des fichiers, nombre de fichiers, ...

La commande finale lancée est :

```
tar --format=posix -W -cf output.tar input_dir
```

Si la commande renvoie un code d'erreur, le statut retourné dans le fichier status est FAILED. Une seule tentative est faite.

Si la commande ne renvoie pas de code d'erreur, le statut retourné dans le fichier status sera COMPLETED.

restart_tar : listes, à une fréquence donnée, des fichiers restarts à archiver.

On utilisera en particulier les options suivantes de la commande tar :

- -W (verify) : lorsque cette option est spécifiée, l'intégralité du tar est vérifiée et comparée au contenu des fichiers sources (octet par octet). Cette option est assez coûteuse puis qu'elle parcourt et lit deux fois les fichiers sources (une fois pour faire le tar, une autre fois pour comparer le contenu).
- --format=posix : qui génnère les tar au format POSIX.1-2001 Ce format supprime les limitations des formats précédents, notamment : longueur des noms de fichiers, taille des fichiers, nombre de fichiers, ...

La commande finale lancée est :

```
tar --format=posix -W -cf output.tar input_dir
```

Si la commande renvoie un code d'erreur, le statut retourné dans le fichier status est FAILED. Une seule tentative est faite.

Si la commande ne renvoie pas de code d'erreur, le statut retourné dans le fichier status sera COMPLETED.

debug_tar : listes, à une fréquence donnée, des fichiers debug à archiver.

On utilisera en particulier les options suivantes de la commande tar :

- -W (verify) : lorsque cette option est spécifiée, l'intégralité du tar est vérifiée et comparée au contenu des fichiers sources (octet par octet). Cette option est assez coûteuse puis qu'elle parcourt et lit deux fois les fichiers sources (une fois pour faire le tar, une autre fois pour comparer le contenu).
- --format=posix : qui génère les tar au format POSIX.1-2001
  Ce format supprime les limitations des formats précédents, notamment : longueur des noms de fichiers, taille des fichiers, nombre de fichiers, ...

La commande finale lancée est :

```
tar --format=posix -W -cf output.tar input_dir
```

Si la commande renvoie un code d'erreur, le statut retourné dans le fichier status est FAILED. Une seule tentative est faite.

Si la commande ne renvoie pas de code d'erreur, le statut retourné dans le fichier status sera COMPLETED.

store_cp : liste des répertoires et fichiers à copier directement sur le CCCSTOREDIR

Une copie sera faite de $INPUT_DMF_DATA vers $OUTPUT_STORE.

La commande finale lancée est :

```
cp –rf  $INPUT_DMF_DATA /input_dir  $OUTPUT_STORE
```

Si la commande renvoie un code d'erreur, le statut retourné dans le fichier status est FAILED. Une seule tentative est faite.

Si la commande ne renvoie pas de code d'erreur, le statut retourné dans le fichier status sera COMPLETED.

work_cp : liste des répertoires et fichiers à copier directement sur le CCCWORKDIR

Une copie sera faite de $INPUT_DMF_DATA vers $OUTPUT_WORK.

Si la commande renvoie un code d'erreur, le statut retourné dans le fichier status est FAILED. Une seule tentative est faite.

Si la commande ne renvoie pas de code d'erreur, le statut retourné dans le fichier status sera COMPLETED.

La commande finale lancée est :

```
cp –rf  $INPUT_DMF_DATA /input_dir  $OUTPUT_WORK
```

launch_ipsl_enlarge.sh

C'est le script shell principal qui fait le lien entre les listes de fichiers à traiter et le script qui réalise les commandes.

Il se lance de la façon suivante :

```
>> ./ launch_ipsl_enlarge.sh
```

On boucle sur les simulations recensées durant l'étape de création de listes dans le fichier et répertoriées dans le fichier $SCRATCHDIR/IGCM_DEM/config.card.liste.

Pour chacune d'entre elles, le script « enlarge_my_files.sh » est lancé.

```
>> ./enlarge_my_files  ${INPUT_DMF_DATA} ${OUTPUT_STORE} ${OUTPUT_WORK}
```

avec

INPUT_DMF_DATA = path de la copie du dmnfs sur l'espace tampon

OUTPUT_STORE=path sur l'espace tampon ou seront stockés les fichiers traités et avant envoi sur CCCSTORE

OUTPUT_WORK=path sur l'espace tampon ou seront stockés les fichiers traités et avant envoi sur CCCWORK

5) Mode d'emploi

Pour traiter les données d'un utilisateur USER, il faut se connecter en tant que USER. Cette version parallèle de l'outil PACK_IPSL, qui réalise des traitements indépendants simultanément, n'a été testé que sur la machine « curie » du TGCC. Il n'y a aucune garantie sur son fonctionnement correct dans un autre contexte.

5.1) Lancement du script de création des listes

- **Récupération de l'outil :**

svn co http://forge.ipsl.jussieu.fr/igcmg/svn/TOOLS/PACK_IPSL PACK_IPSL

- **Les simulations à traiter par utilisateur**

Un fichier doit être donné en argument du la commande ./launch_ipsl_pack.sh. Ce fichier contient les chemins des simulations ou des ensembles de simulations à traiter pour un utilisateur. Ces fichiers seront répertoriés dans le répertoire : ~p86ipsl/PARAM_USER_IPSL sous le nom $USER.txt. La commande à passer pour chaque utilisateur sera alors :

```
./launch_ipsl_pack.sh ~p86ipsl/PARAM_USER_IPSL/$USER.txt
```

- **Les fichiers de configuration**

Dans le répertoire PACK_IPSL, il existe 3 fichiers dits « de configuration ».

Les fichiers de configuration contiennent des variables déterminant les conditions d'exécution du script de traitement des données. Il est donc capital de renseigner ces variables.

- Les répertoires du traitement des données

En tête du fichier « load_ipslPack_env.sh », on trouve des variables contenant les chemins des répertoires utiles au traitement des données :

- « TMP_MIGR_DATA» est le répertoire pour l'établissement préliminaire des listes de fichiers à traiter. **Cette variable est à renseigner.**
- « INPUT_DMF_DATA » est le répertoire contenant les données de l'utilisateur. **Cette variable est à renseigner.**
- « OUTPUT_STORE » et «OUTPUT_WORK» sont les répertoires de stockage des données une fois traitées. Sa structure et les noms de ses sous répertoires sont les même que pour le répertoire de données de l'utilisateur. **Ces variables sont à renseigner.**
- « OUTPUT_CHECK » est un espace qui sert à effectuer les opérations de double vérification. Lui aussi respecte la structure du répertoire de données de l'utilisateur. Cet espace sera par défaut stocké dans $ TMP_MIGR_DATA.
- « OUTPUT_PROGRESS » est un espace conçu à l'origine pour la gestion des reprises en cas de d'arrêt non prévu ou d'échec. Il sert toujours à cela, mais peut aussi être utilisé pour le suivi de l'avancement des opérations de traitements. Cet espace sera par défaut stocké dans $ TMP_MIGR_DATA.

Les fichiers « load_batch_directives.sh » et « loadParameters.sh » ne sont à renseigner que pour l'étape de traitement des données. Voir $5.2.

- **Script de visualisation de l'avancement**

On peut visualiser l'avancement de la création des listes grâce au script « showListsProgress.sh ». Il est indispensable de préciser en argument le même fichier « $USER.txt » utilisé par le script de création des listes.

```
./showListsProgress.sh $USER.txt
```

Il parcourt les simulations de l'utilisateur et pour chaque simulation, donne l'état des listes créées.

Exemple de sortie du script d'avancement :

```
##################################################
###  ETAT D'AVANCEMENT DES SCRIPTS DE lISTES  ###
##################################################

-------------------------------------------------
---------------- Etapes generales ----------------
-------------------------------------------------
find_directory_simul.sh-->OK
create_config_card.sh-->OK

-------------------------------------------------
---------------- Etape createListing -------------
-------------------------------------------------
R55C: createListing.sh-->OK

-------------------------------------------------
---------------  Etape par simu ------------------
-------------------------------------------------
R55C
   calcul_size_simul.sh-->OK
   find_size_pack.sh-->OK
   write_liste_pack.sh-->OK
   archive_restart.sh-->OK
   archive_debug.sh-->OK

=============================================
nbre de simu OK : 1 / 1
=============================================
```

5.2) Lancement du script de traitement « parallèle »

- **Modules à charger**

Avant toute utilisation du script de traitement des données, il faut charger 3 modules :

- Le module « nco » requis pour les opérations de concaténations
- Le module « libccc_user » pour la gestion des arrêts propres des traitements lorsque le temps d'exécution du script est proche de sa limite
- Le module « cdo » pour permettre une double vérification de certains traitements de concaténation

Ces modules sont chargés comme suit :

- `module load nco/4.1.0`
- `module load libccc_user`
- `module load cdo`

Le script de traitement des données opère des vérifications sur ces chargements, et vérifie notamment qu'une version 4.1.0 (ou plus récente) du module « nco » est chargée.

- **Compilation du programme de parallélisation**

Pour cette étape, il faut avoir chargé le module « libccc_user ».

L'outil que l'on compile ici est celui permettant l'exécution simultanée des traitements (parfois appelé abusivement « parallélisation ») appliqués aux données.

Dans le répertoire issu de la base de gestion de configuration, il existe un fichier « makefile » pour effectuer cette compilation.

On tape :

➢ `make`

- **Les fichiers de configuration**

Dans le répertoire issu de la base de gestion de configuration, il existe 3 fichiers dits « de configuration ».

Les fichiers de configuration contiennent des variables déterminant les conditions d'exécution du script de traitement des données. Il est donc capital de renseigner ces variables.

  - <u>Les répertoires du traitement des données</u>

Le fichier « load_ipslPack_env.sh » a du être renseigné avant le lancement du script de création des listes. Voir $5.1.

  - <u>Les directives « batch »</u>

Dans le fichier « load_batch_directives.sh », on trouve des variables qui concernent le fonctionnement « parallèle » du script de traitement des données.

- « nbProcs » est nombre de processeurs mis à contribution pour les traitements. Attention : l'un des processus est maître et organise l'activité des autres : il n'effectue aucun traitement.
- « computationTime » est la durée limite d'exécution du script de traitement des données.
- « timeLimitBeforeEnd » est un intervalle de temps pour l'arrêt propre des traitements avant la fin de l'exécution. Par exemple, si timeLimitBeforeEnd vaut 3600s, aucun nouveau processus ne sera lancé dés que la durée d'exécution du script atteindra une heure avant la date de fin prévue. A partir de cette limite, les processus en cours continueront leur traitement jusqu'à terme.
- Le sens des 3 autres variables (projectName , queueType , QosType) est immédiat, elles doivent être renseignées aussi.

  - <u>Configuration des options</u>

Le fichier « loadParameters.sh » contient les options du script de traitement.

On rappelle ici que le script principal de traitement parallèle des données (« parallelPack.sh ») lance un script secondaire («launch_and_measureTime.sh ») chargé des traitements en parallèle. C'est ce dernier script qui est lancé en « batch ». Il s'auto resoumet lui-même dés que sa durée d'exécution atteint la limite fixée par la variable « computationTime ».
Après chaque exécution de «launch_and_measureTime.sh », il est possible de faire une nouvelle vérification sur certains traitements de concaténation effectués lors cette exécution (ou « instance »). Les traitements de concaténation « doublement vérifiés » sont choisis de manière aléatoire. Par défaut, l'option « double check » est activée et le traitement s'arrête en cas de retour « not OK ». Ainsi :

- « doYouWantCheck » répond à la question : voulez vous une double vérification ? La valeur de cette variable ne peut être que «yes » ou « no ».
- « nbListsToCheck » est le nombre d'opérations de concaténation que l'on souhaite vérifier (si doYouWantCheck=yes).
- « forceRestartFromBegin » : indique au script de recommencer les traitements (des données utilisateur) depuis le début. On explique plus loin que cette option n'est pas prioritaire devant celle indiquée au script directement en ligne de commande.

- **Script de visualisation de l'avancement**

On peut visualiser l'avancement des traitements grâce au script « showPackProgress.sh ».
Utilisé sans option, ce script se lance de la manière suivante :

➢ `./showPackProgress.sh`

Il parcourt les résultats de simulations de l'utilisateur et pour chaque simulation, donne l'état du traitement de chaque liste.
Diverses options sont disponibles :

- -s : pour afficher une simulation en particulier. Il faut fournir le nom de la simulation.
- -o : les sorties de « showPackProgress.sh » sont redirigées vers le fichier donné après cette option.
- --not-detailled : n'affiche que l'état des traitements des simulations, sans le détail des traitements des leurs listes. Ainsi, les simulations sont soit « OK », soit « KO ».

Exemple de sortie du script d'avancement pour une simulation :

```
curie70 - /ccc/cont003/home/dsm/p86caub/SCRIPTS_DEM/TEST_17092012 : ./showPackProgress.sh
specific loading for pack script.
**************************************************
simulation : ARNAUD/IPSLCM5/R55C  ===> not OK
**************************************************
  output_ncrcat :
      R55C_18600101_18691230_1D_histday : COMPLETED at try #1 | time : 41.284
      R55C_18600101_18691230_1M_histmth : COMPLETED at try #1 | time : 9.049
      R55C_18600101_18691230_1M_histrac : COMPLETED at try #1 | time : 5.757
```

```
  output_tar :
    -- no list --
  restart_tar :
      R55C_restart_18600101_18701230 : COMPLETED at try #1 | time : 2.148
  debug_tar :
      R55C_debug_18600101_18700101 : FAILED at try #1 | time : ???
  store_cp :
      cp_files : COMPLETED at try #1 | time : 5.451
  work_cp :
      cp_files : COMPLETED at try #1 | time : 5.06

##################
#####  BILAN #####
##################
nb of Simulations packed with success : 0 / 1
nb of Simus failed : 1
nb of Simus not treated : 0
nb of Lists packed with success : 6 / 7
nb of fails : 1
nb of not treated : 0
nb simul full tared : 0
nb inode before : 359
nb inode after : 293
Total time for elementary operations : 68.749
Time since launch start : 87.581
```

5.3) Exemple d'utilisation

Voici la série de commandes à passer pour traiter certaines données d'entrée du login « bacasable » :

1) Se connecter sur curie

   `svn co http://forge.ipsl.jussieu.fr/igcmg/svn/TOOLS/PACK_IPSL PACK_IPSL`

2) `cd PACK_IPSL`

3) Editer fichier de config : load_ipslPack_env.sh : variables INPUT_DMF_DATA, OUTPUT_STORE, OUTPUT_WORK, TMP_MIGR_DATA

4) Lancement script de listes (ce script est séquentiel et lancé en mode interactif):

5) `./launch_ipsl_pack.sh ~p86ipsl/PARAM_USER_IPSL/bacasable.txt`

(par la suite, pour chaque user il faudra lancer :
./launch_ipsl_pack.sh ~p86ipsl/PARAM_USER_IPSL/$USER.txt)

Le script de liste démarre par le listing des fichiers d'entrée : cela peu prendre du temps avant qu'il ne se passe quoi que ce soit.

6) Visualisation de l'avancement de la génération des listes :

```
./showListsProgress.sh ~p86ipsl/PARAM_USER_IPSL/bacasable.txt
```

En particulier, la ligne « nbre de simu OK » donne une idée de l'état d'avancement.

7) `module load nco/4.1.0 ; module load libccc_user ; module load cdo`

8) `make`

9) Editer le fichier de paramètres : load_batch_directives.sh

10) Lancement script de traitement des données (ce script est parallèle et lance des soumissions en mode batch). A noter que pour un USER, ce script n'est à lancer qu'une fois la création de liste terminée.

```
./parallelPack.sh
```

Par défaut, le script n'efface pas les données d'origine. Pour activer l'effacement, lancer le script avec l'option « -d ».

```
./parallelPack.sh -d
```

11) Visualisation de l'avancement du traitement des données

```
./showPackProgress.sh
```

Sortie dans un fichier output :

```
./showPackProgress.sh –o output_pack.txt
```

5.4) Questions/Réponses (FAQ)

- Comment savoir que le script de création de listes tourne encore ou est terminé ? Que faire en cas de plantage ?

Lancé en mode interactif, le script de création de listes termine son exécution quand il rend la main.
Ce script produit, sur la sortie standard, des messages qui peuvent être utiles pour savoir si son exécution s'est déroulée sans erreur. Pour avoir plus de détails, on peut jouer le script de visualisation de l'avancement de la création des listes de la manière suivante (voir 5.1) :

```
./showListsProgress.sh ~p86ipsl/PARAM_USER_IPSL/bacasable.txt
```

Ce script peut être lance à tout moment de la création des listes (par « launch_ipsl_pack.sh »), y compris lorsque cette phase est terminée. Les différentes étapes de création de listes sont détaillées dans la partie 3).
Le script de suivi de l'avancement de la création des listes affiche le bilan en trois parties, deux générales à l'ensemble des simulations et une par simulation. Dans cette dernière, les échecs sont indiqués en face du nom des sous scripts par des chaînes ' ?? #####'.

- Comment savoir que le script de traitement des données tourne encore ou est terminé ? Que faire en cas de plantage ?

Comme expliqué dans la partie 4), le mode batch est utilisé pour traiter les tâches élémentaires. La commande ccc_mstat ou ccc_mpp indiquera si le job est en queue, en cours ou terminé. En fonction de l'état du job, on pourra réagir comme suit :

- Si un job est encore en train de tourner, la commande `./showPackProgress.sh` permettra de voir l'état du traitement. La chaine de caractère « `still executing...` » confirmera que le travail est en cours.
- Si un job est en queue : une nouvelle instance a été soumise et attend de tourner. La commande `./showPackProgress.sh` permettra de voir l'état du traitement.
- Si le job est terminé (pas de job running ou en queue) :
    - la commande `./showPackProgress.sh` permet de voir l'état du traitement.
    - Le fichier `${OUTPUT_PROGRESS}/SIMULATION_USER/badFailure.txt` donnera l'information sur l'état du traitement : « `Tout s'est fini correctement` », « `il n'y a plus de progression` »,… Si il y a une information de plantage ou pas d'information, relancer le script `./parallelPack.sh`. Si le plantage se reproduit, les fichiers de suivi présents dans le répertoire TRY__* le plus récent, donneront plus d'informations
    - Le monitoring accessible là : http://webservices.ipsl.fr/migration/ donne un compte rendu par utilisateur de ce qui a été traité. Deux cas de figures sont possibles :
        - 100% des simulations (Simu percentage) ont été traitées correctement pour un utilisateur (camembert tout vert). Tout est OK, il n'y a rien à faire.
        - Moins de 100% des simulations (Simu percentage) ont été traitées correctement (il y a du rouge ou du bleu dans le camembert) pour un utilisateur. Il faut alors regarder le rapport des simulations de l'utilisateur (Simu NB de l'utilisateur) et relancer si :

    - il y a au moins une simulation « Not treated »
    - pour une simulation « not OK », en cliquant dessus et en visualisant le détail des listes traitées, il y a au moins un « FAILED » dans la partie « output_ncrcat ».

- Comment reprendre de zéro ?

  Il existe l'option de forçage –f qui permet de repartir de zéro la création des listes et le traitement des données.

```
./launch_ipsl_pack.sh ~p86ipsl/PARAM_USER_IPSL/bacasable.txt –f

./parallelPack.sh –f
```