Wikiprint Book

Title: Public/User_Guide/SCR

Subject: DEEP - Public/User_Guide/SCR

Version: 4

Date: 25.04.2024 09:03:33

Table of Contents

SCR (Scalable Checkpoint Restart)

API

```
/* initialize the SCR library. If not using SIONlib _Buddy_Checkpointing_ use (NULL,NULL) as params, otherwise the check_r
int SCR_Init(int (* check_readable)(char *, void *), void *args);

/* shut down the SCR library */
int SCR_Finalize();

/* determine whether a checkpoint should be taken at the current time */
int SCR_Need_checkpoint(int* flag);

/* inform library that a new checkpoint is starting */
int SCR_Start_checkpoint();

/* determine the path and filename to be used to open a file. Before calling SCR_Start_checkpoint -> delivers path to rest
int SCR_Route_file(const char* name, char* file);

/* inform library that the current checkpoint is complete. valid needs to be provided by user codes. SCR is unable to chec
int SCR_Complete_checkpoint(int valid);

/* register files created without knowledge of SCR, e.g. SIONlib Buddy checkpoints with multiple files. can be retrieved f
* sion_io_stat_t *file_stat; file_stat = sion_get_io_info(sid); before closing the SIONlib file. Then usage is:
* scr_register_files((const char**) file_stat->names, file_stat->nfiles);
* When only using SIONlib to create node-local files _without_ Buddy Checkpointing, this is _NOT_ necessary!
*/
int scr_register_files(const char **filenames, int n);
```

Configuration

SCR can be configured using Environment Variables or via configuration files. Environment variables have to be exported in a batch script via

```
export SCR_VAR=<value>
```

while in configuration files they only need to be defined via

```
SCR_VAR=<value>
```

List of variables

```
SCR_CONF_FILE = <path to conf file> #let SCR use this configuration file
SCR_PREFIX = <directory> # should point to the global file system (typically /sdv-work/$USER/your_checkpoints). All Checkp
SCR_USER_NAME=$USER #mandatory
SCR_JOB_NAME=$PBS_JOBNAME #mandatory
SCR_JOB_ID=`echo  $PBS_JOBID | awk -v FS="." '{print $1}'` #mandatory: used to extract the jobid from PBS for identifying

SCR_COPY_TYPE="FILE"|"LOCAL"|"PARTNER"|"BUDDY"
#"FILE" instructs SCR to use multiple checkpoint descriptors defined in the configuration file
#"SINGLE" instructs SCR to use only _1_ node-local checkpoint
#"PARTNER" instructs SCR using also a partner node to store the checkpoints (SCR intrinsic Buddy-Checkpointing)
#"BUDDY" instructs to use the SIONlib Buddy-Checkpointing. Further information on this can be retrieved from k.thust@fz-ju

SCR_FLUSH=X # Let SCR flush from node-local directory to global file system every X checkpoints
SCR_CACHE_BASE=<cache-directory> # Directory to store node-local files, only used when setting SCR_COPY_TYPE="SINGLE" (def
SCR_CACHE_SIZE=X # Only store X Checkpoints inside the cache directory (default=2)
SCR_FETCH=<1/0> # enable or disable fetching from global file system during restart.
SCR_DEBUG=X # use verbosity level X when running with SCR. Higher X => more output!
```

Configuration File Example

```
SCR_FLUSH=1 # flush every checkpoint
SCR_DEBUG=0 # ne debug output needed
SCR_MODE=PIO #use node-local parallel I/O with e.g SIONlib or HDF5. Other Option is "DEFAULT", to use task-local serial IO
SCR_FLUSH_ASYNC=1 #use asynchronous transfer (only available when using STORE TYPE = BEEGFS



STORE=/mnt/beeond COUNT=10 TYPE=BEEGFS
#using the /mnt/beeond as a cache enables asynchronous transfers of CP files to the global FS (synchronous as well)
#COUNT=X: Store X CPs inside this store
#TYPE=DEFAULT/BEEGFS defines if this store is a BEEGFS Cache or not

#STORE=/tmp COUNT=10 TYPE=DEFAULT #when using DEFAULT, no beegfs flushing will be available. Only synchronous flushing cop

STORE=/tmp #mandatory: Needed to store metadata for SCR

# Specify the different types of checkpoints for a job
# 1 is used to store every time a CP to the BEEGFS Cache in this case
# 2 could be used to create a PARTNER Checkpoint every 4th time.
# 3 could be used to create a BUDDY Checkpoint with SIONlib every 6th time.
# times is by means of calling SCR_Need_checkpoint(int *flag) in your code, when flag=1
# every CKPT descriptor needs an appropiate configured STORE (see above)

CKPT=1 STORE=/mnt/beeond INTERVAL=1 TYPE=SINGLE
#CKPT=2 STORE=/tmp INTERVAL=4 TYPE=PARTNER
#CKPT=3 STORE=/tmp INTERVAL=6 TYPE=BUDDY
```

Buddy-Checkpointing with SIONlib Example This is just an example for writing buddy checkpoints with SIONlib. The Example for reading can be derived from this straight-forward. Do not forget to encapsulate these calls in a proper environment (e.g. call SCR_Start_checkpoint, SCR_route_file… etc. beforehand.)

```
   sion_file_check_par_args_mpi *buddy_args;
  buddy_args = sion_file_check_par_args_init_mpi("br,buddy=1",MPI_COMM_WORLD,1,comm_node);

  SCR_Init(&sion_file_check_par_cb_mpi,buddy_args);

  sion_io_stat_t *file_stat;
  sid = sion_paropen_mpi(path, "bw,buddy=1", &sion_num_files, MPI_COMM_WORLD, &comm_node,        &chunksize,&fsblksize,&si

  saved_data += sion_coll_fwrite(particles, sizeof(particles_block_t), nb, sid);

  // get file stats from sionlib
  file_stat = sion_get_io_info(sid);

  // register files with SCR
  scr_register_files((const char**) file_stat->names, file_stat->nfiles);
  sion_parclose_mpi(sid);
```