FTI - Usermanual

Leonardo Bautista Gomez, Kai Keller

November, 2016

Contents

1 Introduction

In high performance computing (HPC), systems are built from highly reliable components. However, the overall failure rate of supercomputers increases with component count. Nowadays, petascale machines have a mean time between failures (MTBF) measured in hours or days and fault tolerance (FT) is a well-known issue. Long running large applications rely on FT techniques to successfully finish their long executions. Checkpoint/Restart (CR) is a popular technique in which the applications save their state in stable storage, frequently a parallel file system (PFS); upon a failure, the application restarts from the last saved checkpoint. CR is a relatively inexpensive technique in comparison with the process-replication scheme that imposes over 100% of overhead.

However, when a large application is checkpointed, tens of thousands of processes will each write several GBs of data and the total checkpoint size will be in the order of several tens of TBs. Since the I/O bandwidth of supercomputers does not increase at the same speed as computational capabilities, large checkpoints can lead to an I/O bottleneck, which causes up to 25% of overhead in current petascale systems. Post-petascale systems will have a significantly larger number of components and an important amount of memory. This will have an impact on the systemâĂŹs reliability. With a shorter MTBF, those systems may require a higher checkpoint frequency and at the same time they will have significantly larger amounts of data to save. Although the overall failure rate of future post-petascale systems is a common factor to study when designing FT-techniques, another important point to take into account is the pattern of the failures. Indeed, when moving from 90nm to 16nm technology, the soft error rate (SER) is likely to increase significantly, as shown in a recent study from Intel. A recent study by Dong et al. explains how this provides an opportunity for local/global hybrid checkpoint using new technologies such as phase change memories (PCM). Moreover, some hard failures can be tolerated using solid-state-drives (SSD) and cross-node redundancy schemes, such as checkpoint replication or XOR encoding which allows to leverage multi-level checkpointing, as proposed by Moody et al.. Furthermore, Cheng et al. demonstrated that more complex erasure codes such as Reed-Solomon (RS) encoding can be used to further increase the percentage of hard failures tolerated without stressing the PFS.

blabla...

2 Multilevel Checkpointing

2.1 L1

L1 denotes the first safety level in the multilevel checkpointing strategy of FTI. The checkpoint of each process is written on the local SSD of the respective node. This is fast but possesses the drawback, that in case of a data loss and corrupted checkpoint data even in only one node, the execution cannot successfully restarted.

2.2 L2

L2 denotes the second safety level of checkpointing. On initialisation, FTI creates a virtual ring for each group of nodes with user defined size (see ??). The first step of L2 is just a L1 checkpoint. In the second step, the checkpoints are duplicated and the copies stored on the neighbouring node in the group.

That means, in case of a failure and data loss in the nodes, the execution still can be successfully restarted, as long as the data loss does not happen on two neighbouring nodes at the same time.

2.3 L3

L3 denotes the third safety level of checkpointing. In this level, the checkpoint data trunks from each node getting encoded via the Reed-Solomon (RS) erasure code. The implementation in FTI can tolerate the breakdown and data loss in half of the nodes.

In contrast to the safety level L2, in level L3 it is irrelevant which of nodes encounters the failure. The missing data can get reconstructed from the remaining RS-encoded data files.

2.4 L4

L4 denotes the fourth safety level of checkpointing. All the checkpoint files are flushed to the parallel file system (PFS).

3 API Reference

4 Cofiguration

4.1 [Basic]

4.1.1 Head

The checkpointing safety levels L2, L3 and L4 produce additional overhead due to the necessary postprocessing work on the checkpoints. FTI offers the possibility to create an mpi process, called *HEAD*, in which this postprocessing will be accomplished. This allows it for the application processes to continue the execution immediately after the checkpointing.

Value	Meaning
0	The checkpoint postprocessing work is covered by the application processes.
1	The HEAD process accomplishes the checkpoint postprocessing work (notice: In this case, the number of application processes will be (n-1)/node).

(*default = 0*)

4.1.2 Node_size

Lets FTI know, how many processes will run on each node (*npp*). In most cases this will be the amount of processing units within the node (e.g. 2 CPU's/node and 8 cores/CPU $\rightarrow$ 16 processes/node).

Value	Meaning
npp (int $>$ 0)	Number of processing units within each node (notice: The total number of processes must be a multiple of groupsize $\times$ nodesize)

(*default = 2*)

4.1.3 Ckpt_dir

This entry defines the path to the local hard drive on the nodes.

Value	Meaning
string	Path to the local hard drive on the nodes

(*default = _BLANK_*)

4.1.4 Glbl_dir

This entry defines the path to the checkpoint folder on the PFS (L4 checkpoints).

Value	Meaning
string	Path to the checkpoint directory on the PFS (notice: The directory has to be created before execution).

(*default = /path/to/global/storage/*)

4.1.5 Meta_dir

This entry defines the path to the meta files directory. The directory has to be accessable from each node. It keeps files with informations about the topology of the execution.

Value	Meaning
string	Path to the meta files directory. (notice: The directory has to be created before execution).

(*default = /home/username/.fti*)

4.1.6 Ckpt_L1

Here, the user sets the checkpoint frequency of L1 checkpoints.

Value	Meaning
L1 freq. (int ≥ 0)	L1 checkpointing frequency in min^{-1}. If the value is equal to 0, L1 checkpointing is disabled.

(*default = 3*)

4.1.7 Ckpt_L2

Here, the user sets the checkpoint frequency of L2 checkpoints.

Value	Meaning
L2 freq. (int ≥ 0)	L2 checkpointing frequency in min^{-1}. If the value is equal to 0, L2 checkpointing is disabled.

(*default = 5*)

4.1.8 Ckpt_L3

Here, the user sets the checkpoint frequency of L3 checkpoints.

Value	Meaning
L3 freq. (int ≥ 0)	L3 checkpointing frequency in min^{-1}. If the value is equal to 0, L3 checkpointing is disabled.

(*default = 7*)

4.1.9 Ckpt_L4

Here, the user sets the checkpoint frequency of L4 checkpoints.

Value	Meaning
L4 freq. (int ≥ 0)	L4 checkpointing frequency in min^{-1}. If the value is equal to 0, L4 checkpointing is disabled.

(*default = 11*)

4.1.10 Inline_L2

In this entry, the user decides, whether the post-processing work on the L2 checkpoints is done by the HEAD or by the application processes.

Value	Meaning
0	The post-processing work of the L2 checkpoints is done by the HEAD (notice: This setting is only allowed if Head = 1).
1	The post-processing work of the L2 checkpoints is done by the application processes.

(*default = 1*)

4.1.11 Inline_L3

In this entry, the user decides, whether the post-processing work on the L3 checkpoints is done by the HEAD or by the application processes.

Value	Meaning
0	The post-processing work of the L3 checkpoints is done by the HEAD (notice: This setting is only allowed if Head = 1).
1	The post-processing work of the L3 checkpoints is done by the application processes.

(*default = 1*)

4.1.12 Inline_L4

In this entry, the user decides, whether the post-processing work on the L4 checkpoints is done by the HEAD or by the application processes.

Value	Meaning
0	The post-processing work of the L4 checkpoints is done by the HEAD (notice: This setting is only allowed if Head = 1).
1	The post-processing work of the L4 checkpoints is done by the application processes.

(*default = 1*)

4.1.13 keep_last_ckpt

This setting tells FTI whether the last checkpoint taken during the execution will be kept in the case of a successful run or not.

Value	Meaning
0	After `FTI_Finalize()`, the meta files and checkpoints will be removed. No checkpoint data will be kept on the PFS or on the local hard drives of the nodes.
1	After `FTI_Finalize()`, the last checkpoint will be kept (notice: Additionally, the setting *failure* in the configuration file is set to 2. This will lead to a restart from the last checkpoint if the application is executed again).

(*default = 0*)

4.1.14 Group_size

The group size entry sets, how many nodes (members) forming a group.

Value	Meaning
int i ($2 \leq i \leq 32$)	Number of nodes contained in a group (notice: The total number of processes must be a multiple of groupsize $\times$ nodesize).

(*default = 4*)

4.1.15 Verbosity

Sets the verbosity level

Value	Meaning
1	Debug sensitive. Beside warnings, errors and informations, FTI debugging information will be printed.
2	Information sensitive. FTI prints warnings, errors and informations.
3	FTI prints only warnings and errors.
4	FTI prints only errors.

(*default = 2*)

4.2 [Restart]

4.2.1 Failure

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

4.2.2 Exec_ID

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

4.3 [Injection]

4.3.1 rank

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

4.3.2 number

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

4.3.3 position

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

4.3.4 frequenzy

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

4.4 [Advanced]

4.4.1 Block_size

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

4.4.2 Mpi_tag

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

4.4.3 Local_test

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

5 Index

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.