Easy and Efficient Multilevel Checkpointing for Extreme Scale Systems

Leonardo A. BAUTISTA GOMEZ

Barcelona Supercomputing Center (BSC)

July 3, 2018
- Checkpointing in HPC
- Deep Memory Hierarchies
- Multi-level Checkpointing
- Fault Tolerance Interface v1.1
- Future work
- Failures are common in petascale systems
- Long running jobs need to split in multiple runs
- New storage devices, new trade-offs
- C/R is not open, write, close anymore
- Multiple storage levels, multiple resilience levels
- Asynchronism, erasure codes, garbage collection
- 2 x IBM Power9 8335-GTG @ 3.00GHz (20 cores, 160 threads)
- 512GB DRAM
- 2 x Micron 5100 Series 1.9TB SATA SSD
- 2 x Samsung PM1725a 3.2TB NVMe
- 4 x GPU NVIDIA V100 (Volta) with 16GB HBM2
- Single Port Mellanox EDR
- GPFS via one fiber link of 10 GBit
CTE-Power Cluster Configuration
Performance measurements for the SSD and NVMe devices

![Graph showing bandwidth measurements for SSD and NVMe devices across different hostnames. The graph compares read, write, and overall bandwidth performance.]
Random access for WRITE operations (4KB packages)
Random access for READ operations (4KB packages)
Transfer size measurements for the three tested devices

- **GPFS**
  - WRITE: Data points with error bars indicating variance.
  - READ: Data points with error bars indicating variance.

- **SSD**
  - Data points with error bars indicating variance.

- **NVMe**
  - Data points with error bars indicating variance.

The graph shows the bandwidth (MB/s) for different transfer sizes ranging from 4k to 32M bytes, with three devices tested: GPFS, SSD, and NVMe. The y-axis represents bandwidth in MB/s, while the x-axis represents transfer size in bytes.
Weak scaling measurements for the three tested devices

![Graph showing bandwidth vs. nodes for SSD, NVMe, GPFS-1, and GPFS-2 in read and write operations.]
Linear regression for the weak scaling of the SSD and NVMe

- NVME 160p
- SSD 160p
- NVME 64p
- SSD 64p
- Storage Level Abstraction
- Asynchronous Data Transfer
- Erasure Codes and Replication
- Optimal Checkpoint Interval
- Automatic Garbage Collection

```c
#include <stdlib.h>
#include <fti.h>

int main(int argc, char** argv){
    MPI_Init(&argc, &argv);
    char* path = "config.fti"; //config file path
    FTI_Init(path, MPI_COMM_WORLD);
    int world_rank, world_size; //FTI_COMM rank & size
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    int *array = malloc(sizeof(int) * world_size);
    int number = world_rank;
    int i = 0;
    //adding variables to protect
    FTI_Protect(1, &i, 1, FTI_INT);
    FTI_Protect(2, &number, 1, FTI_INT);
    for (; i < 100; i++) {
        FTI_Snapshot();
        MPI_Allgatherv(&number, 1, MPI_INT, array, 1, MPI_INT, FTI_COMM_WORLD);
        number += 1;
    }
    free(array);
    FTI_Finalize();
    MPI_Finalize();
    return 0;
}
```
New FTI Release !!! (https://github.com/leobago/fti/releases)

- 3 I/O modes: POSIX, MPI-IO and SIONlib
- Supported N-to-N and N-to-1 checkpoint modes
- Support for checkpoints with evolving sizes
- Checking checkpoint integrity with MD5 checksums
- Support for Lustre file system
- Integrated Coding with Jerasure 2.0
- Updated developer documentation
- Support for HDF5 checkpoint files
- BSD License
<table>
<thead>
<tr>
<th>Build Jobs</th>
<th>#</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>#726.1</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>#726.2</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>#726.3</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>#726.4</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>#726.5</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>#726.6</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>#726.7</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>#726.8</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>#726.9</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>#726.10</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>#726.11</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>#726.12</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>#726.13</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>#726.14</td>
<td>✔</td>
<td></td>
</tr>
</tbody>
</table>
- Daily visits and frequent questions
- FTI integrated in production codes (GYSELA5D)
- Multiple mini-apps ported (CoMD, CoSP2, LULESH)
• Incremental checkpointing
• Permanent checkpointing
• Integration with batch scheduler
https://github.com/leobago/fti
leonardo.bautista@bsc.es

Thanks