SCR and Preparing for Burst Buffers
DOE COE Performance Portability Meeting

August 23, 2017

Elsa Gonsiorowski
Outline

Burst Buffer Technologies

SCR Overview

Burst Buffers and SCR

Additional Software Projects
# Burst Buffer Technologies

<table>
<thead>
<tr>
<th>Type</th>
<th>Technology</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node Local</td>
<td>IBM BBAPI</td>
<td>LLNL (Sierra)</td>
</tr>
<tr>
<td>Machine Global</td>
<td>Cray Datawarp</td>
<td>LANL (Trinity)</td>
</tr>
</tbody>
</table>
### Burst Buffer Technologies

<table>
<thead>
<tr>
<th>Type</th>
<th>Technology</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node Local</td>
<td>IBM BBAPI</td>
<td>LLNL (Sierra)</td>
</tr>
<tr>
<td>Machine Global</td>
<td>Cray Datawarp</td>
<td>LANL (Trinity)</td>
</tr>
</tbody>
</table>

How can an application utilize this layer for I/O workloads?
Burst Buffers Use Case

- Relies on integration with resource scheduler
- Different for machine-global vs. node-local storage
- Does not address inter-job data movement
Burst Buffers Use Case

Perfect for Checkpoint/Restart
Checkpoint Restart

- a.k.a. Defensive I/O
Checkpoint Restart

- a.k.a. Defensive I/O
- Related to the size of system memory
Checkpoint Restart

- a.k.a. Defensive I/O
- Related to the size of system memory
- Depends on resiliency of machine
Checkpoint Restart

- a.k.a. Defensive I/O
- Related to the size of system memory
- Depends on resiliency of machine
  - Which may change over time
Checkpoint Restart

- a.k.a. Defensive I/O
- Related to the size of system memory
- Depends on resiliency of machine
  - Which may change over time
- Creating a checkpoint may not be as efficient as recomputing
Enable checkpointing applications to take advantage of system storage hierarchies
Enable checkpointing applications to take advantage of system storage hierarchies

- Efficient file movement between storage layers
- Data redundancy operations
SCR Components
SCR Component: Backend Library

- Redirect application files
- Synchronous & asynchronous flush operations
  - Hardware specific capabilities
- Data redundancy
- Support for both checkpoint & output data
int rc = MyApp_Checkpoint(path);
SCR Route file(path, newpath);
int rc = MyApp_Checkpoint(newpath);
SCR Component: Backend Library

SCR_Start_output("dataset name", flags);
SCR_Route_file(path, newpath);
int rc = MyApp_Checkpoint(newpath);
SCR_Complete_output(rc);
SCR Component: Frontend Scripts

- **On Startup** Locate most recent checkpoint and fetch for restart
SCR Component: Frontend Scripts

- **On Startup** Locate most recent checkpoint and fetch for restart
- **Within Allocation** Detect application crash or system failures and trigger restart
SCR Component: Frontend Scripts

- **On Startup** Locate most recent checkpoint and fetch for restart
- **Within Allocation** Detect application crash or system failures and trigger restart
- **During Execution** Manage datasets
SCR Component: Frontend Scripts

- **On Startup** Locate most recent checkpoint and fetch for restart
- **Within Allocation** Detect application crash or system failures and trigger restart
- **During Execution** Manage datasets
- **Resource Scheduler Integration** Pre- and post-stage data movement
SCR Component: Configurations

- Define the levels of the hierarchy
- Define modes/groups of failure
- Define checkpointing and data residency needs
SCR Component: Configurations

- Define the levels of the hierarchy
- Define modes/groups of failure
- Define checkpointing and data residency needs

Machine Portability
Burst Buffers Use Case

Checkpoint Restart
Burst Buffers & SCR: Prestage

- **Machine Global** Solved
  - Global access from CNs to storage
- **Node Local** Requires new softwares
  - Requires deep integration with resource scheduler
  - Most useful for DATs or half+ system jobs
Burst Buffers & SCR: Poststage

- Similar solution for both BB types
- Take advantage of vendor APIs asynchronous operations
- Decouples burst buffer usage from compute usage
  - Requires integration with resource scheduler
  - Allows for more fine-grain control of resources
Unaddressed Concerns

- Applications without checkpointing
- Shared Files
- Arbitrary data movement
  - Machine-learning use case
Combining two codes: FTI and SCR
FTI: variable-based checkpointing scheme
Will support existing FTI and SCR applications
- User-level file system
- Shared namespace across distributed burst buffers
- I/O interception layer
Use parallel processes to perform file operations

- Executed within a job allocation
- dbcast: broadcast from PFS to node-local storage
- dcp: multiple file copy in parallel
- drm: delete files in parallel
- many more

https://github.com/hpc/mpifileutils
SCR Team

https://github.com/llnl/scr

- Kathryn Mohror
- Adam Moody
- Greg Becker
- Elsa Gonsiorowski