Trinity / NERSC-8 Use Case Scenarios

April 5, 2013

This document provides additional views of anticipated usage scenarios for two of the advanced architecture features of Trinity, the burst buffer and power/energy measurement and control. This document may change as our understanding of needs and technologies evolves. These scenarios are not intended to include all intended uses of these technologies.

*Format: Embedded in the **primary** scenario descriptions are numbered tags referring to requirements (listed following each scenario) that are inferred by the scenario. The reader should read the specific requirement, specified by number, at the time it is encountered in each scenario. **Secondary** scenarios (simply labeled scenario) are provided to describe additional capabilities and do not include direct pointers to requirements.*

Burst Buffer Scenarios

Primary Scenario: Checkpoint Restart with Pre-Stage and Drain

A) A new application's latest checkpoint is loaded into the burst buffer to be available for the new job to begin. This may occur before the previous job terminates in order to effectively pre-stage data. (4)(5)(6)

B) The application issues a checkpoint in which data is written in a 'burst' to the burst buffer. Necessary bandwidth is estimated to be in the range of 4.4 - 17.8 TB/s (Trinity)[1] or 2.2 – 8.9 TB/s (NERSC-8)[2]. (2)(7)

   a. Job terminates unexpectedly while writing checkpoint

      i. Partially written checkpoint is removed or eventually overwritten. (6)(8)

      ii. Job restarts using last known good checkpoint and is read from burst buffer, staged from the parallel file system to burst buffer, or read directly from the parallel file system. (4)(5)(6)

      iii. Job begins computing and restarts at step B.

[1] For Trinity, this bandwidth range is estimated based on a JMTTI in the range of 10 – 20 hours and an aggregate memory size in the range of 2 – 4 PB. Based on those ranges and a JMTTI / delta ratio of at least 200 (delta is time to checkpoint 80% of aggregate memory), maximum delta value is in the range of 180 – 360 seconds and minimum burst buffer bandwidth is in the range of 4.4 – 17.8 TB/s.

[2] For NERSC-8, using the same calculation based on an aggregate memory size range of 1 – 2 PB, the corresponding minimum burst buffer bandwidth is in the range of 2.2 – 8.9 TB/s.

C) Every n^{th} checkpoint, or after some time interval, the data is drained off asynchronously to the parallel file system (PFS). The data must be drained fast enough to ensure space is available for the next checkpoint. This performance must be maintained while other applications are writing their checkpoints.[3] Slower drains to the PFS will expose the job to a higher probability of failure without a recent checkpoint. (1)(3)(5)(7)
    a. Failure case – job fails to write to parallel file system
        i. Detection of failure and restart drain from last known good checkpoint or wait for next available checkpoint depending on relative age and time remaining in job. (4)(5)(6)

D) The application writes its final checkpoint to the burst buffer. This last checkpoint is drained to the PFS. The application will not be doing additional computing and the job allocation will soon end. This is similar to step B, except the burst buffer must ultimately clean up any remaining data or pre-stage data for the next job. If the burst buffer in not available or post completion draining is not available, the final checkpoint may be written directly to the PFS. (2)(4)(5)(6)(7)

E) The first application terminates. Cleanup is completed and the next job starts execution. If post completion drain is available, it will continue after the next job has started. (6)(8)

Requirements for Checkpoint Restart with Pre-Stage and Drain

1. The primary requirement is that incorporation of the burst buffer into the system design should be cost-justified by the increase in expected workload productivity in the face of expected levels of system hardware and software failures (including failures of the burst buffer itself).
2. Checkpoint data must be written to the burst buffer rapidly enough to reduce the time required to complete a checkpoint (as compared to the PFS).
3. Selected checkpoints must be copied to the PFS asynchronously to application execution.
4. Checkpoint data is available from the burst buffer for restart within an existing or immediately scheduled new job after partial system failure or to the extent practical, after burst buffer failure.
5. Checkpoint data is available from the PFS for restart on a new job.
6. Checkpoint data can be staged into the burst buffer prior to job start and also drained (written to the PFS) after job completion. This requirement may be satisfied after initial delivery.
7. Burst buffer speed and bandwidth estimates must be based on hardware and file system that are at least 80% full and have undergone a realistic history of file writes and deletions.

---

[3] Future burst buffer designs will always require data to be staged on the burst buffer in order to restart so that a slower parallel file system could ultimately be provisioned.

8. This document does not attempt to prescribe specific behavior for handling burst buffer errors. Rather, hardware or software failures in the use of the burst buffer should be handled in a manner that provides the greatest practical opportunity to continue job execution, perhaps in a degraded mode that bypasses the burst buffer.

Burst Buffer Secondary Scenarios

The remaining Burst Buffer scenarios describe additional capability beyond checkpoint-restart. It would be beneficial to take advantage of such cases, but they must still adhere to the burst buffer requirements, particularly (1)(8). They do not, however, directly address the other requirements as the primary scenario does.

Scenario: Data Cache – Common Read Only Files – Demand Load

This scenario is motivated by poor performance experienced when every compute node attempts to read the same shared object libraries or configuration data files.

A) Each compute node issues a read for a common read only file, typically a shared object library, a configuration file, or a data file that is shared by many or all nodes of the workload.
B) The read is recognized as referring to a file in the data cache, perhaps by its location in the file system name space.
C) The file is loaded into the burst buffer from the backing file system.
D) The compute node's read is satisfied from the burst buffer copy.
E) The same burst buffer copy satisfies reads from additional compute nodes.
F) The burst buffer copy is deleted on job termination or to free up space for a more recent read request.

Scenario: Data Cache – Common Read Only Files – Data Staged

This scenario is similar to the demand load case, except the files are loaded into the burst buffer prior to job start.

A) At job submission time, a list of files is made available to the job scheduler
B) In a manner similar to pre-launch staging of checkpoints, when the time for job start approaches, the listed files are copied into the burst buffers associated with all the nodes of the future job.
C) After job start, each compute node issues a read for a common read only file, typically a shared object library, a configuration file, or a data file that is shared by many or all nodes of the workload.
D) The read is recognized as referring to a file in the data cache, perhaps by its location in the file system name space.
E) The compute node's read is satisfied from the burst buffer copy.
F) The burst buffer copy is deleted on job termination.

Scenario: Temporary Job Data – Out-of-Core and other kinds of working space

This scenario is motivated by workloads that process more required data than main memory can hold. Typically in such cases the data is not present before the job is run and does not need to be stored after the run, thus all they require is an

allocation of burst buffer resources. Such workloads are typically optimized to fetch data from physical disks and may be improved upon with fast access to a large amount of fast burst buffer storage.

A) A paging space could be implemented on burst buffer. Paging strategies are well understood and can likely be used with this, but may decrease the life of SSDs. Technology to monitor and possibly limit a particular job's impact on life of SSDs is implicitly required if the burst buffer is used as paging space.
B) Streaming algorithms may allow for multiple passes on huge volumes of data, especially if the burst buffer is attached to additional compute resources.

Data Analysis and Visualization Secondary Scenarios

The following scenarios are motivated by the opportunity to use the burst buffer to perform analysis and visualization tasks that have not previously been possible. This section covers the entire category of data analysis, including reductions, feature extraction, statistical analysis, compression and visualization. Note that visualization is a subset of data analysis. Data analysis workloads may also make use of the *Data Cache* and *Temporary Data* use cases described previously.
The three burst buffer use cases discussed below are in transit, post-processing and ensemble visualization, and are differentiated from each other by where the data resides prior to being read into the burst buffer.

Scenario: Data Analysis and Visualization: In Transit

This scenario includes analyzing data coming from the compute node after it is generated by simulation, but before it is written to disk. Data arrive from the compute nodes, just after they have been generated. These may be part of the checkpoint process, or may have been reduced on the compute node in situ, for example, as a visualization dump or a statistical reduction. Here are some forms this might take:

A) In transit analysis/visualization of simulation. Data are written to the burst buffer from the compute nodes, as part of a checkpoint or in the form of visualization dumps. The data are analyzed o r visualized on the burst buffer and written to disk.
B) In transit analysis/visualization, followed by spillover. Data are written to the burst buffer from the compute nodes, partially analyzed or reduced, then the product is written to other resources, such as visualization nodes for further processing.
C) Comparative analysis and visualization. Data are written to the burst buffer from the compute nodes, and compared immediately to data from simulation or experiment that had been previously stored and is now read in to the burst buffer for the purpose. The incoming data might also be compared to data from previous time-steps still on the burst buffer.

Scenario: Data Analysis and Visualization: Post-processing– Read/Write Files – Data Staged

This scenario includes analyzing data from simulations that have been stored on the file system. This scenario is similar to the demand load case, except the files are

loaded into the burst buffer prior to job start. Typically each task or process addresses its own share of the data exclusively.

A) At job submission time, a list of files is made available to the job scheduler
B) In a manner similar to pre-launch staging of checkpoints, when the time for job start approaches, the listed files are copied into the burst buffers associated with all the nodes of the future job.
C) After job start, each compute or visualization node issues a read for its file, typically a data file that is associated with the process on that node. For example, a segment of a parallel database.
D) The read is recognized as referring to a file in the data cache, perhaps by its location in the file system name space.
E) The compute/visualization node's reads and writes are satisfied from the burst buffer copy. Typically these read and write requests are not contiguous, with frequent seeks.
F) The burst buffer copy is migrated back to disk on job termination.

For visualization, this process will include geometry extraction from data, and may include on-platform rendering as well as other forms of analysis. This is the traditional form of visual analysis, and for scientific reasons, will remain part of the workflow.

It will also be possible to read many time-steps of reduced visualization data into the burst buffer from the file system, and analyze these as an ensemble across time steps, providing a new visualization capability.

Scenario: Data Analysis and Visualization: Ensembles of Data.

This scenario includes the analysis of large ensembles of data from related simulations. This may take place in transit, as the data from the simulations are written to the burst buffer, or in post-processing, as datasets are read from the file system, or as a mixture of both.

A) Ensemble visualization. Data is written to the burst buffer from the compute nodes, as part of a checkpoint or in the form of visualization dumps. Data may also be read from the file system. Data from different simulations are compared.

Notes on the burst buffer in terms of visualization use

- Visualization use would depend on both reads and writes from the burst buffer.
- If the burst buffer is implemented on a node with processing capability, it could be useful to do reduction, analysis or visualization on the burst buffer itself.
- Reductions may be saved on the burst buffer for further analysis, to be compared with future time-steps or with data already saved to disk. Sufficient memory would be useful for doing comparisons of small parts of data with incoming data.
- Reductions may also be sent to a special-purpose rendering machine or to remote machines, via the LAN or DISCOM, so reasonable connectivity to those resources would be useful.

- Reductions and analysis products may be saved to disk, as in the checkpoint case.
- The burst buffer would be shared between the existing checkpoints and the analysis data and products, so attention to scheduling or partitioning of the burst buffer will be needed. How this is done would depend on the size of the reduced analysis data relative to that of the checkpoint data.
- Visualization may be batch or interactive. Interactive visualization is inherently bursty, so this would also impact scheduling schemes.

Power/Energy Measurement and Control Scenarios

Primary Scenario: Increase Application Efficiency

Discussion: Application Efficiency will be determined by an as yet undefined metric that will likely include performance, energy, priority of job, amortized node expense and time-of-day (to list a few) as variable and/or weighted parts of eventual fused metric (goodness value). In concept, this is similar to Energy Delay Product (EDP) but more inclusive and targeted to a particular sites needs.
This scenario, for the purposes of simplification, will focus on performance (wall clock execution time) and energy (combined total energy used by a single application on all nodes used by the application for the duration of application execution).

A) Execute a large-scale production scientific computing application (application) using the default system environment and parameters. This first run is a productive run that produces productive results. This run also establishes the baseline energy and performance characteristics of this application (1) (Note: other factors might affect the application such as scale and input problem requiring separate or additional analysis).
B) Analyze the energy and performance data to determine what available tuning parameters might be applied to follow on executions (2).
C) Execute SAME application (keep as many factors, number of nodes, problem type etc. the same as possible) with tuning parameters applied (3). The energy and performance results from this run will be judged relative to the baseline execution (1,2). Note, this is also a productive run producing useful production results.
D) Analyze (reference (B), 2).
E) Additional component level analysis to determine if additional tuning can be productively applied (4).
F) Additional executions of the SAME application until range of productive tuning parameters are established (1,2,3,4).

Notes on Increasing Application Efficiency:

This scenario focuses on applying tuning parameters to hardware power management capabilities, e.g. affecting the frequency and subsequent energy use of the CPU. This could be accomplished in a number of ways including setting P, C or S states, or any other architectural mechanism exposed for this purpose. We value any opportunity to affect energy used by ANY component if it can be leveraged to increase the energy efficiency of our applications, and/or to operate the system within its externally allocated and variable power budget. The cycle outlined is a general high-level scenario. The process is repeated using production applications so all time consumed for analysis is the result of productive runs. The output of the analysis is an understanding of the effect on performance and energy of tuning parameters that can be applied to this application at this scale for this problem. The knowledge gained by this analysis can be used to simply run the application more efficiently (using the proposed fused metric for example) or to implement intelligent power capping of the overall platform (described in a separate scenario).

Requirements for Increasing Application Efficiency:

1) To obtain an energy profile, the amount of energy used by a single application on all nodes used by the application, the minimum requirement is a node level measurement capability. Since an application will be executed on a large number of nodes and there is no expectation that these nodes will coincide perfectly with a set of cabinets a node level measurement capability, rather than cabinet level, is required. The frequency of data sampling per node should be greater than or equal to, one sample per second to obtain enough fidelity for analysis and comparison with subsequent runs. It is expected that at the node level the data samples will be DC measurements, discrete current and voltage values.
2) This analysis step implies that the measurement data be made available for analysis, at a minimum, after application execution. Also implied is a transport mechanism to coalesce the data to a single location. The transfer of data from the points of measurement must scalable and efficient to be of utility. Further, this also implies tools are available for analysis such as generating a energy total for all nodes involved in the application and possibly visualization capabilities to analyze characteristics of the energy profile of the application over the duration of the run. These tools will help determine the most productive tuning parameters to apply to subsequent executions. An out-of-band tightly integrated Reliability Availability and Serviceability (RAS) subsystem could be leveraged to accommodate many of these capabilities.
3) To affect the energy used, tuning parameters must be exposed, for example, to the OS, run-time, launch mechanism, scheduler or application library. For example, before application execution all cores of all nodes that will host the application are set to a lower frequency state. After execution, the nodes are reset to default values. This implies an ability to control CPU frequency, for example. This is more of a static approach to CPU frequency tuning.
4) Initial tuning parameters may be applied statically. Multiple static tuning configurations may be applied and analyzed based solely on composite or

node level analysis. It may be determined that to achieve additional application energy efficiency, or to achieve any relative to baseline, dynamic tuning methods must be applied. Component level measurement is required to determine where energy is being used within a node. For example, intense compute, communication or IO phases can be observed with component level measurement. Dynamic tuning can be applied to allow the CPU to run at very high frequency during computationally intensive phases. During heavy communication or IO phases the CPU can be set at lower energy saving frequencies. Other components can be tuned if the capability is available and exposed.

Primary Scenario: Power Capping

Discussion: Power Capping, minimally defined as the ability to proscribe the instantaneous power draw, energy use (over time) or power/energy fluctuation (including rate of change and magnitude). This scenario will suggest that two methods of Power Capping should be applied in conjunction to maximize the use of the underlying resource while protecting against accidental violations of Power Cap parameters. This scenario will assume that the facility manager has defined, for example, the sites power, energy and cooling parameters for the period of time covered in this scenario and the Platform manager has defined any other parameters that will be used to define Power Capping levels such as other local policy considerations used in the fused metric defined in Scenario 1. The two approaches are: 1) hardware Power Cap – defined by setting a physical limit to the amount of instantaneous power that the platform, cabinet, node or component is limited to, or rate of change limitation, 2) Power aware scheduling – defined as intelligently scheduling jobs to maintain a mix of power consumption (or energy use) that complies with site policies.

A) System Administrators set platform Power Caps as directed by Facility and Platform management (1).
B) Users schedule applications (over a period of time) that have been previously analyzed (as in Scenario 1)(2).
C) Scheduler launches applications with tuning parameters necessary to keep overall platform within Power Cap parameters (3).
D) Scheduler does not adequately launch application mix to maintain power/energy use within Power Cap parameters (4).

Notes Power Capping:

This does not account for a dynamically changing Power Cap, jobs launched based on what caps were at the time of launch.

Requirements for Power Capping:

1) This activity requires an integrated ability to configure the platform as a whole, at the cabinet, node and or component level to gate the power and or energy consumption at a hardware level. Efficiently configuring the platform

to accomplish this implies that this is an activity that can be accomplished while the platform is operational (it will not be practical to only have the option of accomplishing this configuration at boot time for example). As described in Requirement 2 - Scenario 1, a RAS system could be leveraged to efficiently accomplish this activity.

2) Implies there is a way for either the user to specify the range of power/energy tuning parameters and associated profile information or this information is available by some other means to the scheduler.
3) In addition to the requirements described in Scenario 1, which enable individual applications to be analyzed from a power and performance perspective, this activity requires an ability for a scheduler to use the specific power and energy characteristics and tuning parameters as part of the fused metric calculation mentioned in Scenario 1 to determine how (what tuning parameters) and when (when this application given the known power/energy profile) each application can be scheduled to run to most efficiently use the resource while remaining within the proscribed Power Cap parameters.
4) This implies that the power parameters configured in step (A) act as a fail-safe preventing the platform, cabinet, node or component from violating the Power Cap parameters.