Lawrence Livermore National Laboratory

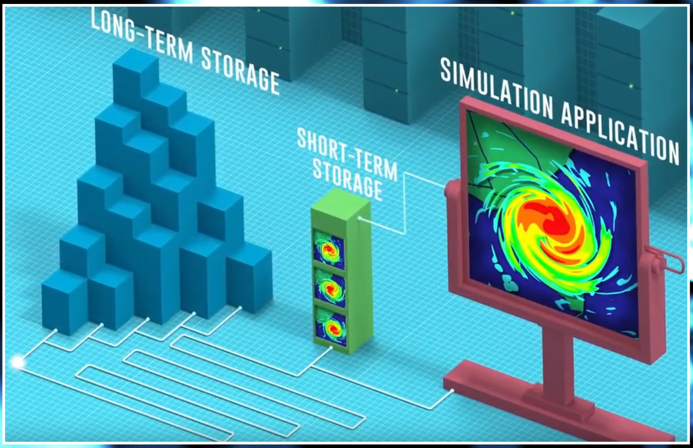
7000 East Avenue
Livermore CA 94550

Contacts
Adam Moody
Kathryn Mohror
Franck Cappello

SCR Framework: Accelerating Resilience and I/O for Supercomputing Applications

Prepared for:

2019 R&D 100 Award Entry



LLNL-MI-776537

Prepared by LLNL under Contract DE-AC52-07NA27344.

SCR Framework: Accelerating Resilience and I/O for Supercomputing Applications

1. PRODUCT/SERVICES CATEGORIES

A. Title

SCR Framework: Accelerating Resilience and I/O for Supercomputing Applications

B. Product Catagory

Software/Services

2. R&D 100 PRODUCT/SERVICE DETAILS

A. Primary submitting organization

Lawrence Livermore National Laboratory

B. Co-developing organizations

Argonne National Laboratory

C. Product brand name

The Scalable Checkpoint/Restart Framework 2.0 (SCR)

D. Product Introduction

This product was introduced to the market between January 1, 2018, and March 31, 2019.

This product is not subject to regulatory approval.

E. Price in U.S. Dollars

Open source

F. Short description

The Scalable Checkpoint/Restart Framework 2.0 (SCR) enables high performance computing simulations to take advantage of hierarchical storage systems, without complex code modifications. With SCR, scientific simulations' input/output performance can be improved by orders of magnitude, with their results produced in significantly less time than they could be with traditional methods.

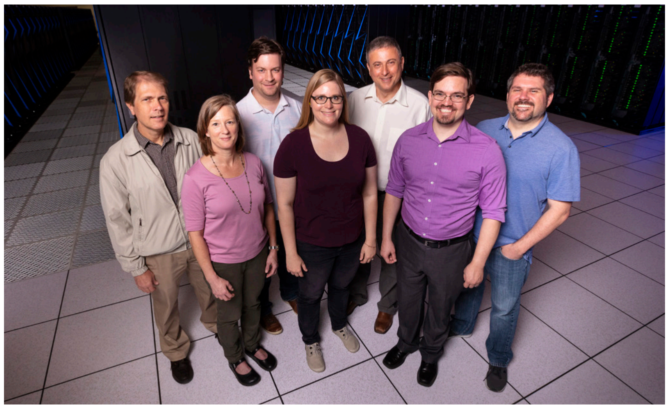
G. Type of institution represented

Government or independent lab/institute

H. Submitter's relationship to product

Product developer

I. Photos



(Left to right) Bronis R. de Supinski, Kathryn Mohror, Tony Hutter, Elsa Gonsiorowski, Greg Kosinovsky, Cameron Stanavige, and Adam Moody.

J. Video

https://youtu.be/_r6svl_eAns

3. PRODUCT/SERVICE DESCRIPTION

A. What does the product or technology do?

The Scalable Checkpoint/Restart Framework 2.0 (SCR) enables high performance computing (HPC) science applications to perform input/output (I/O) operations and manage resilience orders of magnitude faster than they could with traditional methods. Version 2.0 was released in March 2019, and SCR now includes I/O management support in addition to the resilience features that have supported HPC applications for about a decade. SCR utilizes fast storage tiers on HPC supercomputers to quickly cache application and resilience data so that applications can produce their results in less time. This is important because HPC applications simulate real-world phenomena that impact our daily lives. For example, the results of models that predict the behavior of hurricanes (see Figure 1) are used by policy makers to make important decisions such as where and when to evacuate. Scientific simulations can typically take hours, days, or even weeks to compute. Delays in getting the results of these simulations due to I/O slowdowns or failures can have catastrophic impacts, including the possible loss of human lives.

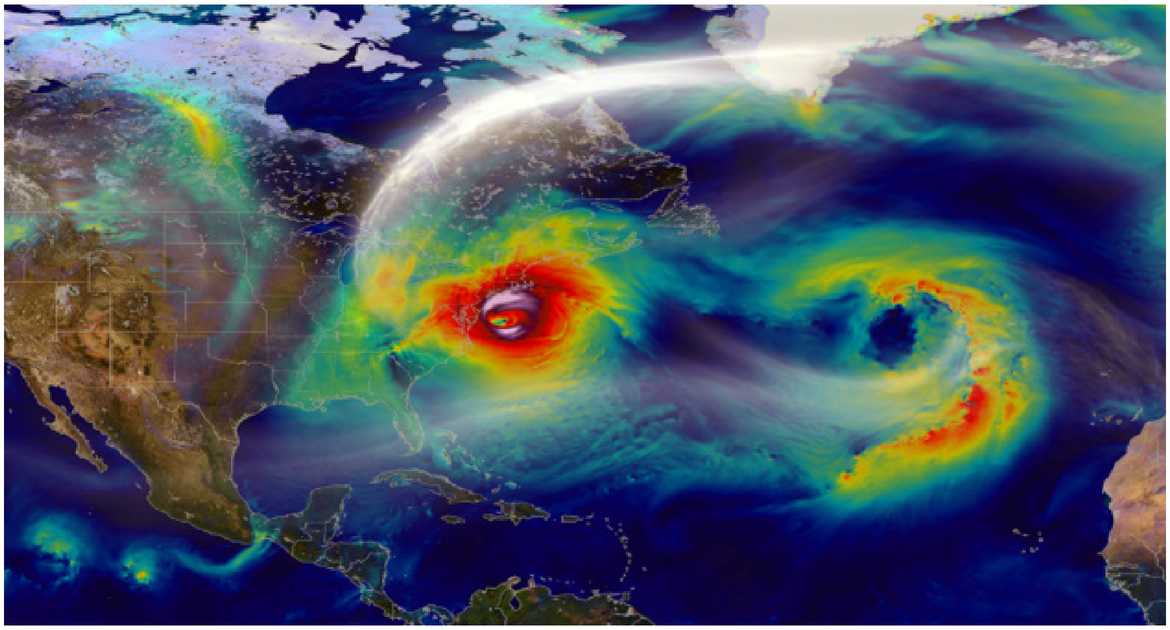


Figure 1: Output image from a simulation of Hurricane Sandy. The results of simulations like this are needed as quickly as possible to save lives and property. The new I/O management features in the Scalable Checkpoint/Restart Framework 2.0 (SCR) enable applications to output finer-grained results faster than ever before. (Courtesy of NASA: https://www.nas.nasa.gov/SC13/demos/demo21.html#prettyPhoto[pp_gal]/1/)

HPC and SCR

HPC, the applied use of supercomputers, has become critically important for industry and government R&D efforts, engineering applications, and scientific discovery, with wide-reaching applications, including product design and development (e.g., advances such as additive manufacturing), precision medicine, and modeling of complex systems (e.g., Earth's climate). HPC has been cited as an essential component of national economic competitiveness and even national security, leading to initiatives such as the Department of Energy's Exascale Computing Project as part of the National Strategic Computing Initiative in the United States, the Partnership for Advanced Computing in Europe, and substantial investments in HPC by Japan and China. Furthermore, systems research in HPC has been increasingly adopted by the general computing market (e.g., processors with higher numbers of compute cores and greater adoption of HPC programming technologies such as MPI and OpenMP).

Over time, HPC supercomputers have increased in computing power largely by increasing in scale (i.e., by increasing the node or processor core count). The larger scale enables scientists to simulate larger and more complex problems or to compute existing problems much faster. Unfortunately, the larger the scale of the HPC application run, the greater the challenges for I/O and resilience. With traditional methods, documented cases report applications losing as much as half their execution time to I/O operations and resilience—meaning it can take twice as long for the application to complete and deliver its results owing to the slowness of these operations. However, with SCR, I/O and resilience activities are managed efficiently and transparently so that users can get their time-critical results without prohibitive delays or significant code changes.

Why Is I/O Slow on HPC Systems?

Performing I/O operations to a parallel file system (a large, reliable, semipermanent storage space that is shared across all jobs running on a cluster and sometimes across several clusters in a compute center) on a supercomputer is expensive, in terms of time taken, at large scale, where a single output operation can take on the order of tens of minutes (Ross et al. 2006, Iskra et al. 2008), and some input operations have been reported to take on the order of hours (Frings et al. 2013). It is important to note that during I/O operations performed in the traditional way without SCR, HPC applications are blocked waiting on the I/O operations to complete—which means no computational progress is made toward computing the solution. Furthermore, the current trend is that computational capabilities of large-scale supercomputers increase more quickly than I/O capabilities, so the I/O performance promises to get worse instead of better with new supercomputers.

Why Is Resilience Challenging on HPC Systems?

Although supercomputers are made of high-quality components, the systems become less reliable at larger scales simply because there is a higher probability that any of the larger number of components could malfunction or outright fail at any time. Long-running scientific applications typically encounter mean times between failures on the order of hours owing to hardware or software breakdowns (Schroeder et al. 2006, Gupta et al. 2017) and soft errors (Michalak et al. 2005). The common way for applications to tolerate these failures is with checkpoint/restart, a process by which the application periodically saves its state to checkpoint files on reliable storage, typically a parallel file system. Should a failure occur, the application can restart from a prior state by reading a checkpoint file.

Unfortunately, although checkpoint/restart is a tried-and-true approach to resilience on HPC systems, checkpointing operations performed in the traditional way without SCR suffer from the same I/O bottlenecks as all other I/O operations, and applications can spend tens of minutes or more blocked, waiting on I/O operations to complete during each checkpoint cycle. The impact of this is significant loss of computing time and delay in producing results. For example, if an application produces output every hour that takes 10 minutes to complete when writing to the parallel file system alone, in a 24-hour allocation the application will have lost 4 hours due to I/O operations. Additionally, restarting is a manual and time-consuming process without SCR. When a failure occurs, the application stops and the resource manager removes the application from its reserved allocation. That means that the application user must frequently check the application's progress to see whether a failure has occurred; if one has, the user must manually resubmit the job in the resource manager queue to restart it, with hours or even days passing before the run starts again.

How Does SCR Help with I/O Bottlenecks and Resilience?

SCR addresses the problems of I/O bottlenecks and resilience in HPC applications by utilizing fast, intermediate levels of storage on HPC systems in addition to the traditional parallel file system and providing a robust and full-featured checkpoint/restart scripting infrastructure to aid in failure recovery. The utilization of intermediate levels of storage by SCR enables applications to achieve high-performance I/O without the need for complex and non-portable code changes. SCR's scripting infrastructure automatically manages many of the manual and time-consuming tasks required by HPC users in monitoring and restarting jobs in the event of failures. We show the major features of SCR in Figure 2.

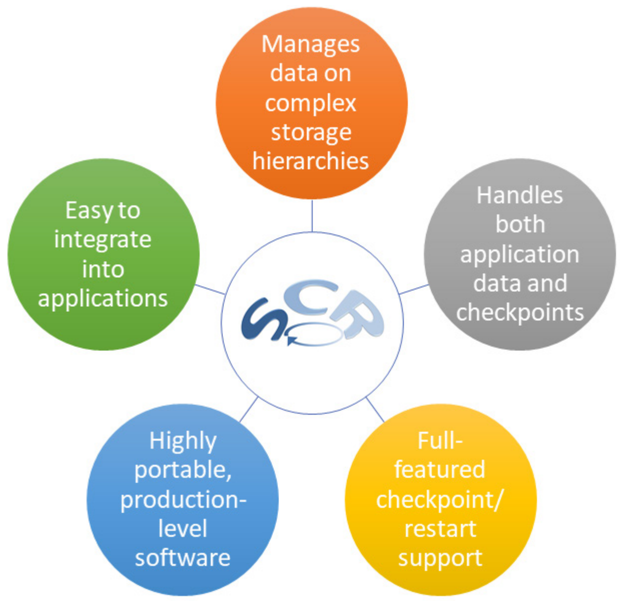

Manages data on complex storage hierarchies
Handles both application data and checkpoints
Full-featured checkpoint/restart support
Highly portable, production-level software
Easy to integrate into applications
SCR

Figure 2: SCR provides important features for enabling high performance computing (HPC) applications to use hierarchical storage systems effectively. It transparently manages application data on complex storage hierarchies for portable HPC applications. SCR also handles both general application data and checkpoint data using scalable data management strategies. The full-featured checkpoint/restart support in SCR greatly eases the burden on users of monitoring and restarting applications. SCR is highly portable and production-level software that runs on a variety of systems and storage hierarchies. Finally, the SCR abstract programming interface integrates easily into HPC applications as a simple wrapper around existing input/output code.

The intermediate levels of storage SCR utilizes are a relatively new addition to HPC systems and can consist of main memory on the compute nodes (RAM disk) or solid-state drives (SSDs) located on the compute nodes or possibly on I/O nodes (sometimes called “burst buffers”), or any other type of storage that might exist on the system. These intermediate levels of storage provide temporary locations that can be used to mitigate high I/O overhead because they are not shared across the whole supercomputer and suffer from little to no contention compared to the parallel file system.

While intermediate levels of storage represent an incredible performance opportunity, they are very challenging for applications to utilize directly. This challenge comes primarily from the fact that every HPC system has a unique storage hierarchy composed of different storage devices with different abstract programming interfaces (APIs) for moving data between storage levels. All these factors mean that for an application to achieve high-performance I/O on hierarchical storage, the developer must undertake a huge system-specific coding effort, which could result in fragile and non-portable code. However, if an application is integrated with SCR, SCR transparently manages the application’s data and checkpoints on differing storage hierarchies without any application code changes.

The checkpoint/restart scripting infrastructure manages all aspects of resilience for application users, including checking compute node health, detecting application hangs or failures, and restarting the application when needed. These features can improve the resilience performance of HPC applications by orders of magnitude. Additionally, SCR removes the need for application users to frequently check their application runs to ensure they have not failed and to manually restart the runs if they do fail.

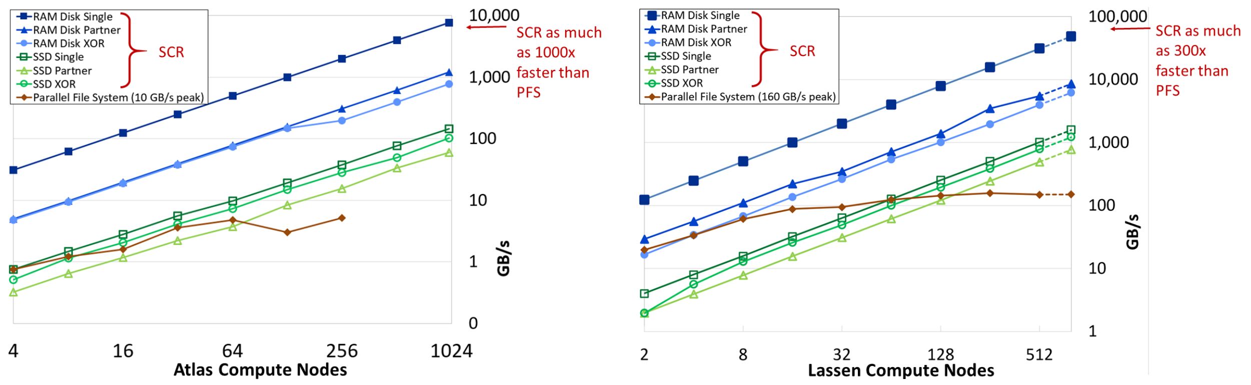


Figure 3: SCR's input/output (I/O) techniques scale linearly with the number of compute nodes in the scientific application run, as we show for two different supercomputers, Atlas (left) and Lassen (right). In contrast, the parallel file system reaches its scaling limit at about 64 compute nodes. SCR's I/O mechanisms are as much as 1,000× faster than the parallel file system alone, saving scientific applications valuable time in computing their results.

In Figure 3, we show the scaling performance of I/O operations using SCR. In the figure, Single, Partner, and XOR refer to protection schemes that SCR employs to protect data on intermediate storage. In each case, the techniques applied by SCR scale linearly with the number of compute nodes in the job and can outperform the parallel file system. We show results on two Lawrence Livermore National Laboratory supercomputers, Atlas and Lassen, where SCR's mechanisms on temporary storage outperform the parallel file system when used alone by as much as 1,000× on Atlas and 300× on Lassen. In practice, SCR uses a combination of protection schemes in a single application run, and we have measured I/O improvement of as much as 234× in a real run of a laser–plasma interaction code.

What Is the Impact of SCR?

The first version of the Scalable Checkpoint/Restart Framework was in production use by HPC applications for more than a decade and supported a wide variety of HPC platforms, resource managers, and hierarchical storage designs. A new feature was added in the Scalable Checkpoint/Restart Framework 2.0 (now known as SCR), which was released in March 2019; SCR now provides support for I/O management on hierarchical storage systems. This new functionality in SCR enables application users to output more detailed and frequent data from their application runs, which can result in a more comprehensive understanding of the scientific output.

SCR addresses the challenges of I/O management on hierarchical storage and resilience with three critical features: it (1) provides an easy-to-use API for applications to use hierarchical storage for I/O, (2) manages and protects users' data on tiered storage with high performance, and (3) supports a full set of features to facilitate checkpoint/restart for HPC applications. The API is portable across systems and enables applications to take full advantage of hierarchical storage for both application and checkpoint data, with only minor code modifications.

B. How does the product operate?

SCR Architecture and Overview of Functionality

SCR consists of three major components that support high-performance I/O and resilience on hierarchical storage systems (see Figure 4). First, to achieve high portability, SCR is highly configurable via environment variables and configuration files. Second, the SCR library that is linked with an HPC application implements the SCR API and manages the complexity of storing and moving data through the storage hierarchy. Third, SCR provides full-featured checkpoint/restart support through a set of scripts that interact with the resource manager and computing environment.

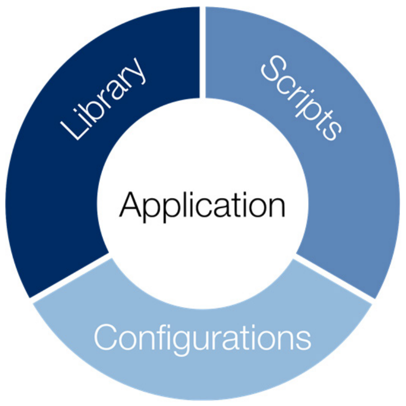


Figure 4: SCR consists of a library that links with a high performance computing (HPC) application and a set of scripts that interact with the HPC resource manager. Additionally, SCR is highly configurable via environment variables or configuration files so that it is portable to different HPC storage hierarchies and so it meets the needs of the job currently running.

The components of SCR work in harmony to support the efficient execution of large-scale HPC applications. In Figure 5, we show the major functionality of each of the components during an application run. SCR performs a large number of functions to help HPC applications run efficiently on supercomputers. In the figure, the two green circles represent user activities and the rest of the circles represent activities performed by SCR on behalf of the user: brown circles represent activities performed by the SCR scripting environment, and orange circles are performed by the SCR library linked into the simulation application (red circle). Light blue arrows represent activities that are performed before and after execution of the HPC simulation job. The dark blue arrows show activities that occur during the simulation application run.

Overall, SCR performs a large number of complex tasks for HPC users, including performing health checks of the computing environment, monitoring the progress of the scientific application, managing output data and checkpoint files with high performance, and making sure final data is transferred to the parallel file system. Without SCR, users would need to implement complicated, system- and application-specific code to support all the functionality that SCR manages.

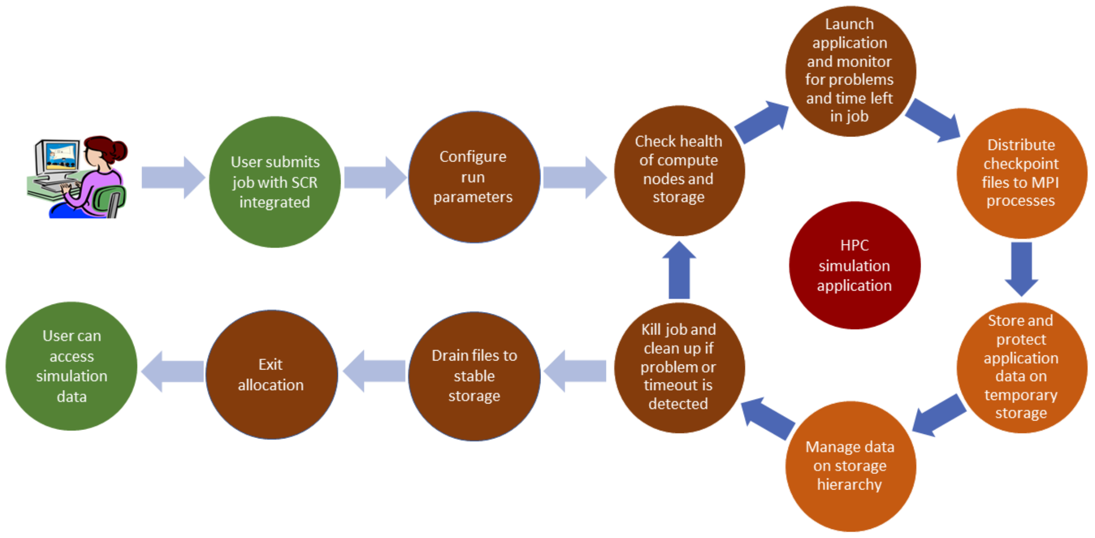

User submits job with SCR integrated
Configure run parameters
Check health of compute nodes and storage
Launch application and monitor for problems and time left in job
Distribute checkpoint files to MPI processes
HPC simulation application
Store and protect application data on temporary storage
Manage data on storage hierarchy
Kill job and clean up if problem or timeout is detected
Drain files to stable storage
Exit allocation
User can access simulation data

Figure 5: SCR performs a large number of functions to help high performance computing (HPC) applications run efficiently on supercomputers. Green circles represent user activities; brown circles represent activities performed by the SCR scripting environment; and orange circles are performed by the SCR library linked into the simulation application (red circle). Light blue arrows represent activities that are performed before and after execution of the HPC simulation job. The dark blue arrows show activities that occur during the simulation application run.

Support for Arbitrary Storage and Protection Levels

A key feature of SCR is that it is highly portable to a variety of HPC hierarchical storage systems because of its flexible configuration strategy. For example, for each storage device, system administrators can specify the size of the device, how many checkpoints should be stored on the device before an old one is deleted, and its failure characteristics. This information enables SCR to protect user data on temporary storage with high confidence. As another example, users can optionally specify the storage location to use as the first level of storage, the protection mechanisms to use at each level of storage, and how much time SCR should allow for draining the last checkpoint and data files to the parallel file system before the end of the job allocation.

SCR also offers flexible and portable protection mechanisms for files stored on temporary storage locations. The mechanisms vary in cost (i.e., how much storage is required and how long it takes to complete the protection mechanism) and resilience (i.e., the kind of failures the mechanism protects against). As an example, the simplest, fastest, and least resilient mechanism is called Single, through which checkpoint or application data is stored on the first level of storage on the compute node, usually a RAM disk or SSD. This is fast but not very resilient because if a compute node fails, the checkpoint or application data is lost since there is no backup. Another example of a protection mechanism is Partner, where a copy of the application data or checkpoint is stored on a partner compute node. This has a higher cost because now two copies of each file are stored on the first level of storage and transferring the copy to the partner compute node takes time. Now, if a compute node is lost because of failure, there is a backup copy of the data on the partner compute node. Users can configure SCR to manage and protect their data on a variety of hierarchical storage systems with different devices and using an arbitrary (and configurable) number of protection levels.

Easy-to-Integrate API for Managing Application Data and Checkpoints

Another important feature of SCR is that the SCR API is easy for application developers to integrate into their existing I/O or checkpoint code. In Figure 6, we show an example of writing checkpoint data using pseudocode. The developer simply needs to add three API calls around the existing I/O code. The SCR start and complete API calls tell SCR when an output phase is started and completed. The flag “checkpoint” supplied to the SCR start routine indicates that the data to be written will be checkpoint data. The SCR route call replaces the original file name with a new file name, which enables SCR to direct the application to write to the first level of storage instead of to the parallel file system. To have SCR manage application data, the developer simply needs to specify “output” for the flag to the SCR start API call. SCR supports data sets that are both checkpoint data and application data if the user specifies both flags in the SCR start API call.

```
function checkpoint (my_data, filename) {



  open file filename
  write file my_data
  close file


  return
}
```

```
function checkpoint (my_data, filename) {

  call SCR_Start_output(CHECKPOINT)
  call SCR_Route_file(filename, scr_file)

  open file scr_file
  write file my_data
  close file

  call SCR_Complete_output()
  return
}
```

Figure 6: The SCR abstract programming interface (API) is easy to integrate into complex applications because it is a simple wrapper around existing application input/output (I/O) code. In this example, we show pseudocode for a checkpointing function in an application. On the left is the original code, where the application opens a file called "filename," writes data to it, then closes it. To use SCR to write data, the application developer simply needs to insert three API calls that wrap the existing I/O code.

SCR Provides Full-Featured Support for Checkpoint/Restart

The final feature of SCR that we highlight here is the many complex tasks that SCR undertakes to support checkpoint/restart, including detecting and restarting from failures. To achieve this functionality, the SCR scripts are integrated with resource managers commonly found on high performance computing systems. The scripts compartmentalize resource manager–specific handling of common tasks. For example, the mechanism to detect a hung job varies across systems, so SCR abstracts this functionality within its scripts. SCR manages many tasks in support of checkpoint/restart, including maintaining a full history of checkpoints previously saved to the parallel file system and automatically reloading the most recent checkpoint or any other checkpoint as directed by the user at the start of each job; detecting and avoiding reloading corrupted checkpoints; monitoring the application and detecting failures or application hangs and automatically killing and restarting any problematic runs in the same allocation; and terminating the application before the end of the allocation with enough time to drain the last checkpoint and any output to the parallel file system.

C. Product Comparison

SCR has several features that put it far above existing competitive products. Namely, it supports an arbitrary number of storage and protection levels—defined by users or system administrators—for application I/O, which provides flexibility for application characteristics and system design; requires little code modification to integrate into applications; has full-featured support for checkpoint/restart via integration with resource managers; produces small checkpoint sizes; and supports application I/O as well as checkpoint I/O. Additionally, it is of production quality and has been used in production application runs since 2007 and has been ported to a variety of systems.

Here, we compare SCR to three state-of-the-art products that are the closest match to SCR with respect to functionality:

• The Fault Tolerance Interface (FTI): https://github.com/leobago/fti
• The Very Low Overhead Checkpointing System (VeloC): https://github.com/ECP-VeloC/VELOC
• Distributed MultiThreaded CheckPointing (DMTCP): http://dmtcp.sourceforge.net/

Arbitrary storage and protection levels: SCR supports an arbitrary number of storage and protection levels, using a simple configuration file. This means that SCR is highly portable to differing storage hierarchies and can easily take advantage of new or different storage hardware resources that are installed on high performance computing systems. In contrast, FTI and VeloC do not support an arbitrary number of levels but have a fixed number, which makes them less flexible for application users and limits system portability. DMTCP supports a single level of checkpointing (the parallel file system).

Low code modification requirement: The code modifications required by SCR are relatively low. To use SCR, applications simply need to insert wrapper calls around their existing checkpoint/restart or I/O code. In our experience with applications over the years, the modifications needed to integrate SCR are relatively minor and can be done by a person who has little expertise with SCR. In contrast, FTI requires that application developers annotate every variable in the code that needs to be saved, which can be a significant undertaking for a large legacy code. VeloC has two interfaces: one that is similar to that of FTI and that requires marking every variable; and one that is similar to SCR that wraps existing checkpoint/restart code. The usage of the SCR-like API in VeloC is similar in complexity to using SCR, but VeloC is not yet production ready. DMTCP is a system-level checkpointing system, which means that no application changes are required in order to obtain checkpoints.

Full-featured checkpoint/restart support: SCR boasts sophisticated integration with the resource managers on high performance computing systems. The SCR scripting tools provide automatic restart of applications in the event of failure, provide a health check of resources like storage devices and compute nodes to make sure that application runs have the best chance of succeeding, and detect when the current allocation is nearing its end so they can automatically stop the application execution and drain the last checkpoint to the parallel file system so it is available for the next allocation. FTI does not have system integration with resource managers. The VeloC checkpointing system does have some integration with resource managers but is not as sophisticated as SCR, and VeloC is not yet a production-ready system. And while DMTCP has integrated with some resource managers to facilitate launching and restart, its integration is limited and it does not support advanced features such as health checks or automatic restart.

Modest checkpoint sizes: SCR, FTI, and VeloC are application-level checkpointing libraries, which means that the data saved in checkpoints is only what is needed for restarting. Smaller checkpoints take less time and resources to store them and can be saved in fast tier storage like RAM disk. In contrast, system-level checkpointing libraries like DMTCP save all system state needed for restart, which means that the checkpoints can be very large and take more resources for storage as well as have longer checkpoint and restart times. System-level checkpointing has the advantage of not requiring any code modifications at the expense of large checkpoint files and limited portability. Additionally, system-level checkpointing libraries are less portable because they require significant work to be integrated on different operating systems.

Supports general I/O as well as checkpoint I/O: The Scalable Checkpoint/Restart Framework was updated to version 2.0 (now known as SCR) and released in March 2019, and it supports handling general data files with similar mechanisms as for handling checkpoint files. This means that users can employ the fast storage tiers of high performance computing systems for their large output files without needing to implement complex, non-portable code to support a system's storage hierarchy. The output files of an application will be automatically moved to the parallel file system asynchronously so the application can go back to computing its results. In contrast, no other checkpointing system supports the handling of general output. With the competition, users must write their output directly to the parallel file system or implement the complex code themselves. Further, if an output set is lost before it is copied to the parallel file system because of catastrophic system failure, SCR will ensure the application is restarted using the most recent checkpoint saved just before the output was written so that it will be generated again.

Production quality and portable: SCR has been in production use for more than a decade and supports a variety of systems. SCR supports several resource manager configurations, including SLURM, MOAB, ALPS, and LSF. It supports several vendor-supplied burst buffer APIs, including Cray DataWarp and IBM BB API. Because it is an application-level checkpointing library, it does not need to be integrated with operating system features but is independent of operating systems unlike DMTCP. The other application-level checkpointing libraries, FTI and VeloC, are not implemented at the same production level as SCR. FTI is a research prototype and supports Linux and Cray systems. VeloC, while targeted to support upcoming exascale high performance computing platforms, is still under development and is not at the same production level as SCR. VeloC currently supports Linux systems and BB API.

D. Comparison summary

Table 1: SCR boasts many features above the competition. Each row represents the features supported by SCR and tools that are similar to SCR. Beneath the features, we use a green check to indicate "full support," a yellow check to indicate "partial support," and a red cross to indicate "no support."

System	Features					Production quality	Portability
	Number of checkpoint levels supported	Code modifications required	Integration with system resources	Checkpoint size	I/O management		
SCR	🟩✓ (Arbitrary)	🟩✓ (Low; requires wrappers around existing code)	🟩✓ (Resource manager integration: LSF, SLURM, ALPS)	🟩✓ (Saves only what is needed by the application)	🟩✓ (Can manage general output files)	🟩✓ (In production use since 2007)	🟩✓ (Supports Linux, Cray)
Fault Tolerance Interface (FTI)	🟨✓ (Supports 4 storage levels)	🟥✗ (Requires marking all variables)	🟥✗ (Does not provide system integration)	🟩✓ (Saves only what is needed by the application)	🟥✗ (Supports only checkpoint output)	🟨✓ (Limited use in production)	🟩✓ (Supports Linux, Cray)
VeloC	🟨✓ (Supports 3 storage levels)	🟩✓ (Low; requires wrappers around existing code)	🟩✓ (Resource manager integration: LSF, SLURM)	🟩✓ (Saves only what is needed by the application)	🟥✗ (Supports only checkpoint output)	🟨✓ (Still in early development)	🟨✓ (Supports Linux)
DMTCP	🟥✗ (Single-level support)	🟩✓ (No code modifications)	🟩✓ (Resource manager integration: SLURM, Torque)	🟥✗ (Saves all needed system state)	🟥✗ (Supports only checkpoint output)	🟩✓ (In production use since 2004)	🟨✓ (Supports Linux)

E. Limitations

- Does not support shared files written by applications [can be integrated with external software products for this support (e.g., Unify File System)]
- Assumes a globally coordinated checkpointing model that is common in high performance computing applications
- Requires some code modifications to use
- Requires some system knowledge to install [but can be done by a knowledgeable person (system administrator) for all users]

4. SUMMARY

The Scalable Checkpoint/Restart Framework 2.0 (SCR) enables high performance computing (HPC) simulations to take advantage of hierarchical storage systems, without complex code modifications. Version 2.0 was released in March 2019, and SCR now includes input/output (I/O) management support in addition to the resilience features that have supported HPC applications for about a decade. SCR utilizes fast storage tiers on HPC supercomputers to quickly cache application and resilience data so that applications can perform I/O operations orders of magnitude more quickly than they could with traditional methods and produce their results in less time. SCR has several features that support I/O and resilience for HPC applications that put it far above existing competitive products, including requiring few code modifications, providing full-featured support for checkpoint/restart, and managing general application data in addition to checkpoint data. The lightweight data management strategies employed by SCR are as much as 1,000× faster than the parallel file system alone and have been demonstrated to improve the I/O performance of large-scale production application by 234×.

5. CONTACT INFORMATION

Principal investigator from each of the submitting organizations:

Adam Moody
Computer Scientist
Lawrence Livermore National Laboratory
925-422-9006
moody20@llnl.gov

Kathryn Mohror
CASC Data Analysis Group Leader
Lawrence Livermore National Laboratory
925-423-2997
kathryn@llnl.gov

Franck Cappello
Senior Computer Scientist
Argonne National Laboratory
630-252-0715
cappello@anl.gov

Full list of development team:

Elsa Gonsiorowski
Systems Software Developer
Lawrence Livermore National Laboratory
925-422-1127
gonsiorowski1@llnl.gov

Cameron Stanavige
Software Engineer
Lawrence Livermore National Laboratory
925-422-4379
stanavige1@llnl.gov

Bronis R. de Supinski
Chief Technology Officer
for Livermore Computing
Lawrence Livermore National Laboratory
925-422-1062
bronis@llnl.gov

Gregory Becker
Computer Scientist
Lawrence Livermore National Laboratory
925-422-7445
becker33@llnl.gov

Kathleen Shoga
Computer Scientist
Lawrence Livermore National Laboratory
925-422-5850
shoga1@llnl.gov

Greg Kosinovsky
Software Engineer
Lawrence Livermore National Laboratory
925-423-0130
kosinovsky1@llnl.gov

Tony Hutter
Software Developer
Lawrence Livermore National Laboratory
925-424-3556
hutter2@llnl.gov

Tanzima Islam
Postdoctoral Researcher
Former Lawrence Livermore National Laboratory Employee
360-650-3601
islamt2@wwu.edu

Kento Sato
Postdoctoral Researcher
Former Lawrence Livermore National
Laboratory Employee
+81-78-940-5840
kento.sato@riken.jp

Raghunath Raja Chandrasekar
Doctoral Candidate/Student Employee
Former Lawrence Livermore National
Laboratory Employee
614-364-5080
rajachan@protonmail.com

Greg Bronevetsky
Computer Scientist
Former Lawrence Livermore National
Laboratory Employee
607-279-3485
bronevet@google.com

Rinku Gupta
Software Developer
Argonne National Laboratory
630-252-6266
rgupta@anl.gov

Bogdan Nicolae
Computer Scientist
Argonne National Laboratory
630-252-1969
bogdan.nicolae@acm.org

Sheng Di
Assistant Computer Scientist
Argonne National Laboratory
630-252-1520
sdi1@anl.gov

Media and public relations person who will interact with R&D's editors regarding entry material:

Connie Pitcock
Business Development and Marketing Associate
Lawrence Livermore National Laboratory
925-422-1072
pitcock1@llnl.gov

Person who will handle banquet arrangements for winners:

Linda Becker
CASC Division Secratary
Lawrence Livermore National Laboratory
925-423-0421
becker22@llnl.gov

6. AFFIRMATION

By submitting this entry to R&D Magazine you affirm that all information submitted as a part of, or supplemental to, this entry is a fair and accurate representation of this product. You affirm that you have read the instructions and entry notes and agree to the rules specified in those sections.
For more information, please call 973-920-7032 or email rdeditors@advantagemedia.com

7. REFERENCES

W. Frings, D. H. Ahn, M. LeGendre, T. Gamblin, B. R. de Supinski & F. Wolf. 2013. “Massively Parallel Loading.” In Proceedings of the 27th international ACM conference on International conference on supercomputing, pp. 389–98

S. Gupta, T. Patel, C. Engelmann & D. Tiwari. 2017. “Failures in Large Scale Systems: Long-Term Measurement, Analysis, and Implications.” In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC ‘17)

K. Iskra, J. W. Romein, K. Yoshii & P. Beckman. 2008. “ZOID: I/O-Forwarding Infrastructure for Petascale Architectures.” In Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP ’08), pp. 153–62

S. E. Michalak, K. W. Harris, N. W. Hengartner, B. E. Takala & S. A. Wender. 2005. “Predicting the Number of Fatal Soft Errors in Los Alamos National Laboratory’s ASC Q Supercomputer.” In IEEE Transactions on Device and Materials Reliability, vol. 5, no. 3, pp. 329–35

R. Ross, J. Moreira, K. Cupps & W. Pfeiffer. 2006. “Parallel I/O on the IBM Blue Gene/L System.” In Blue Gene/L Consortium Quarterly Newsletter, Tech. Rep., First Quarter

B. Schroeder & G. A. Gibson. 2006. “A Large-Scale Study of Failures in High-Performance Computing Systems.” In Proceedings of the International Conference on Dependable Systems and Networks (DSN), June, pp. 249–58