Award ER25750: Coordinated Infrastructure for Fault Tolerance Systems
Indiana University Final Report

Investigator: Andrew Lumsdaine
Center for Research in Extreme Scale Technologies,
Indiana University
E-mail: lums@cs.indiana.edu
2719 E. 10th Street Bloomington, IN 47408

March 8, 2013

Contents

1 Project Information

- Title: Coordinated infrastructure for Fault Tolerant Systems (CiFTS)
- Principal Investigator: Andrew Lumsdaine
- Award Number: ER25750
- Organization: CREST, Indiana University.
- Submitted By: Andrew Lumsdaine, Principal Investigator

2 Project Participants

- Andrew Lumsdaine, Principal Investigator
- Joshua Hursey, Ph.D. student and Postdoctoral Research Scholar
- Abhishek Kulkarni, Ph.D. student
- DongInn Kim, Research staff member
- Lizhe Wang, Research staff member

3 Executive Summary

The main purpose of the Coordinated Infrastructure for Fault Tolerance in Systems initiative has been to conduct research with a goal of providing *end-to-end fault tolerance on a systemwide basis for applications and other system software*. While fault tolerance has been an integral part of most high-performance computing (HPC) system software developed over the past decade, it has been treated mostly as a collection of isolated stovepipes. Visibility and response to faults has typically been limited to the particular hardware and software subsystems in which they are initially observed. Little fault information is shared across subsystems, allowing little flexibility or control on a system-wide basis, making it practically impossible to provide cohesive end-to-end fault tolerance in support of scientific applications.

As an example, consider faults such as communication link failures that can be seen by a network library but are not directly visible to the job scheduler, or consider faults related to node failures that can be detected by system monitoring software but are not inherently visible to the resource manager. If information about such faults could be shared by the network libraries or monitoring software, then other system software, such as a resource manager or job scheduler, could ensure that failed nodes or failed network links were excluded from further job allocations and that further diagnosis could be performed.

As a founding member and one of the lead developers of the Open MPI project, our efforts over the course of this project have been focused on making Open MPI more robust to failures by supporting various fault tolerance techniques, and using fault information exchange and coordination between MPI and the HPC system software stack–from the application, numeric libraries,

and programming language runtime to other common system components such as jobs schedulers, resource managers, and monitoring tools.

3.1 Research Approach

With the Coordinated Infrastructure for Fault Tolerance Systems (CIFTS, as the original project came to be called) project, our aim has been to understand and tackle the following broad research questions, the answers to which will help the HPC community analyze and shape the direction of research in the field of fault tolerance and resiliency on future high-end leadership systems.

- Will availability of global fault information, obtained by fault information exchange between the different HPC software on a system, allow individual system software to better detect, diagnose, and adaptively respond to faults? If fault-awareness is raised throughout the system through fault information exchange, is it possible to get all system software working together to provide a more comprehensive end-to-end fault management on the system?
- What are the missing fault-tolerance features that widely used HPC system software lacks today that would inhibit such software from taking advantage of system-wide global fault information?
- What are the practical limitations of a system-wide approach for end-to-end fault management based on fault awareness and coordination?
- What mechanisms, tools and technologies are needed to bring about fault awareness and coordination of responses in a leadership-class system?
- What standards, outreach and community interaction are needed for adoption of the concept of fault awareness and coordination for fault management on future systems?

Keeping our overall objectives in mind, the CIFTS team has taken a fourfold approach.

- Our central goal was to design and implement a *light-weight, scalable infrastructure* with a *simple, standardized interface* to allow communication of fault-related information through the system and facilitate coordinated responses.

  This work led to the development of the Fault Tolerance Backplane (FTB) publish-subscribe API specification, together with a reference implementation and several experimental implementations on top of existing publish-subscribe tools.

- We enhanced the intrinsic fault tolerance capabilities representative implementations of a variety of key HPC software subsystems and integrated them with the FTB.

  Targeted software subsystems included: MPI communication libraries, checkpoint/restart libraries, resource managers and job schedulers, and system monitoring tools.

- Leveraging the aforementioned infrastructure, as well as developing and utilizing additional tools, we have examined issues associated with expanded, end-to-end fault response from both system and application viewpoints.

From the standpoint of system operations, we have investigated log and root cause analysis, anomaly detection and fault prediction, and generalized notification mechanisms. Our applications work has included libraries for fault-tolerance linear algebra, application frameworks for coupled multiphysics applications, and external frameworks to support the monitoring and response for general applications.

- Our final goal was to engage the high-performance computing community to increase awareness of tools and issues around coordinated end-to-end fault management.

  Our outreach activities covered a broad spectrum, including technical papers and presentations, demonstrations, numerous community-oriented discussion venues, hosting of students as summer interns, and interactions with HPC vendors.

3.2 Research Accomplishments

Our objective as part of the CIFTS team was to design and construct a set of tools and components that will replace the current, deficient, fault-paradigms used on high-performance computers with a robust system for the next generation of leadership-class computers. Our work in this area was driven by six objectives: design and create a fault backplane that can coordinate the fault response for both system and user components, implement a variety of fault tolerance capabilities into Open MPI, investigate extensions to MPI for fault tolerance, integrate Open MPI with the fault backplane, integrate a wide range of key libraries and applications into the system, and deploy the enhanced software on the DOEs largest computers in support of their national mission. The accomplishments during the course of this project in comparison to the original goals and objectives are listed below:

1. The widely deployed implementation of the Message Passing Interface (MPI) standard, Open MPI, was enhanced to integrate with the FTB, and support a common set of fault-related events. We have defined basic interfaces between MPI and FTB with the help of discussions with the tools and applications community. We augmented the Open MPI runtime system to exchange fault information with other system components. The interface and complete specification was also extended to include support for watchdog/autonomic components to help in fault prediction and prevention.

2. The use of coordinated checkpoint restart with BLCR in Open MPI was established as the primary fault tolerance mechanism. This effort has also helped in specifying additions and modifications to BLCR and the partners respective MPI implementations that were required for checkpoint/restart integration in the MPI implementations. We prototyped integration of incremental checkpointing, differential checkpointing and process migration with Open MPI. The addition of uncoordinated checkpoint support using message as a team effort has helped make Open MPI more robust in the event of fail-stop failures.

3. Error and abort reporting in Open MPI was extended to catch and respond to signals from network fabric indicating network failure, path migration, and topology change. This admitted us to prototype adaptive collective communication algorithms that reroute communication around broken or low-performing communication links.

4. A series of public releases of the FTB API specification, beginning with version 0.5 in June 2008, and culminating in the draft version 1.0 specification.

   The FTB API specifications have been accompanied by releases of the reference implementation of the FTB. The FTB implementation works in IBM Blue Gene, Cray, and Linux cluster environments, and is released under the BSD license.

5. A variety of system monitoring tools and libraries have been FTB-enabled to make hardware fault information available via the FTB, including the Reliability, Availability and Serviceability (RAS) systems of Cray and IBM Blue Gene systems, the Intelligent Platform Management Interface (IPMI), Ganglia, and Syslog.

6. We have made significant advances in the ability of system operators to navigate, understand, analyze, and act upon the large volumes of RAS log data that is often associated with larger supercomputer installations.

7. Leveraging the FTB and other capabilities developed within the project, we have demonstrated novel application-level resilience capabilities. One example integrates specific services and capabilities within a framework for coupled multiphysics simulations, while the other provides a an external "Fault Correlation Framework" (FCF) which can provide monitoring and resilience services for generic applications with little or no modifications.

4 Technical Approach

From a technical perspective, the CIFTS framework consists of the FTB API specification and the software that use this FTB API, as shown in Figure 1. The FTB API can be used by system software ranging from operating systems and job schedulers to math libraries, file systems, and high-level user applications. In addition to existing software, third-party developers can set up automatic scripts, diagnostic routines, fault-information analysis engines, and logging systems that can be FTB-enabled to communicate with other FTB-enabled software.

4.1 The FTB API Specification

The FTB API is a publish-subscribe framework that describes the interface that can be used by any HEC system software to publish and obtain fault information from the system. The FTB API interface consists of a dozen routines that allow system software to connect to and disconnect from the FTB, publish fault events, and subscribe and unsubscribe to these fault events based on a set of filters. As an example: the FTB API provides a routine called *FTB Connect* to be used by every FTB client to initialize itself and connect to the FTB system. The FTB client must specify various details including the namespace in which it plans to publish its events. Namespace is an important concept in the FTB API specification. The FTB API specification imposes no restrictions on the fault information that an FTB client can publish. While the FTB API specification provides the interface to publish/subscribe to fault events, the semantics of the fault events are independent of FTB API specification and must be understood and defined by a software prior to using the FTB interface. To this end, the FTB API incorporates an event namespace, portions of which

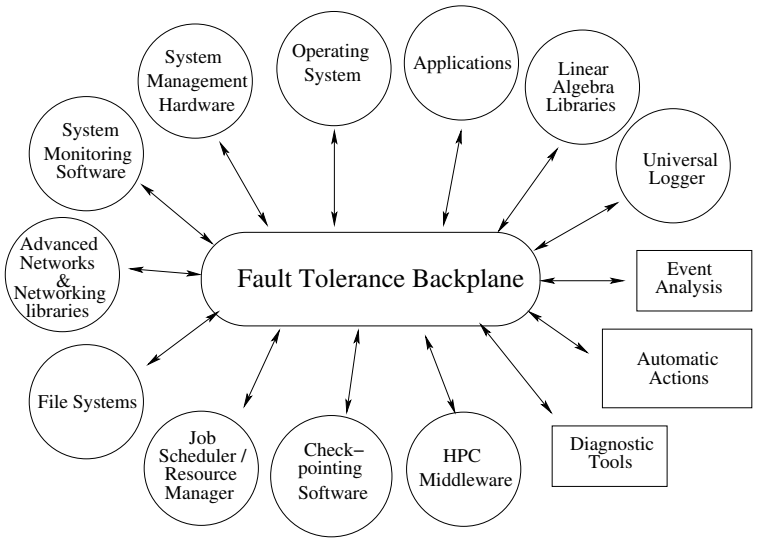

Figure 1: CIFTS Framework

are reserved for the different FTB-enabled software programs. In the FTB framework, prior to publishing any event the FTB client must specify the namespace where it plans to publish its fault events. Similarly, FTB clients wishing to receive events need to ensure that they have registered their interest to receive events in the correct namespace.

The CIFTS team released the FTB API version 0.5 in June 2008. Based on community and vendor feedback, we are currently working on the FTB API version 1.0. Versions of the FTB API specification can be found on the CIFTS website [21].

4.2 The FTB software - The CIFTS FTB API Implementation

The "FTB software" is an implementation of the FTB API specification developed by the CIFTS team. The FTB software was first publicly released in Sept. 2008. Currently based on the FTB API version 0.5, the FTB can be viewed as an asynchronous messaging backplane that allows communication of fault events among the different HEC software systems.

The FTB physical infrastructure is based on a distributed architecture, as shown in Figure 2. The FTB framework comprises a set of distributed daemons, called as FTB agents. These agents incorporate the bulk of the FTB logic and manage the bookkeeping as well as communication of events throughout the FTB system.

The FTB agents, on startup, connect and organize themselves into a tree-based topology. The initial topology construction takes place with the assistance of the FTB bootstrap server which provides information that helps every FTB agent determine its parent FTB agent and position in the topology tree. During its lifetime, if an agent loses its parent, it can connect itself (and its children and its attached FTB clients) to a new parent in the topology tree, making the topology tree self-healing with a certain level of fault tolerance. The bootstrap server can also be made fault tolerant to a certain extent by keeping track of the topology information and specifying redundant bootstrap servers. The FTB agents subsequently connect to the existing agent topology tree when

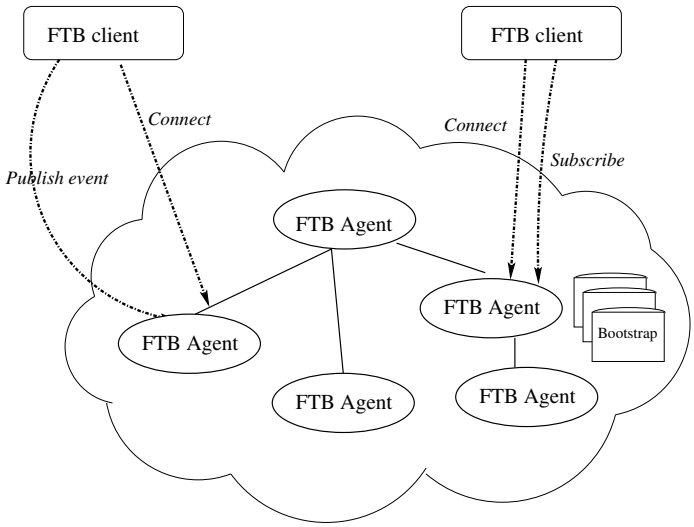

Figure 2: The FTB Architecture

they startup. The FTB clients, on startup, connect to a local FTB agent by using FTB routines (as described in the FTB API specification). Alternatively, in the absence of a local FTB agent, the FTB client connects to a remote FTB agent by enlisting the assistance of the FTB bootstrap server. Once a connection is established, the FTB client can publish events and subscribe to receive events using the FTB Client API.

The FTB agents keep track of all registered FTB clients. The agents also keep track of all FTB client subscription requests, along with the subscription criteria. They perform incoming event matching against subscription criteria and send events to the correct destinations and clients. In addition, they keep track of their tree topology and metadata associated with maintaining connections and routing information. In summary, the majority of the FTB logic lies with the FTB agent. Further details about the design of the FTB implementation can be found in [36].

Details on CIFTS and the FTB software implementation have been a focus of several talks and presentations [28–35] given during this project.

4.3 Improving Fault Tolerance in Open MPI

The Message Passing Interface (MPI) is one of the most important programming models in high-performance computing. MPICH2, [23], Open MPI [45], and MVAPICH2 [44] are three of the most popular MPI implementations that heavily dominate the high-performance computing space [19]. Our focus in the project was on the following areas:

1. Standardizing faults information and conditions under which these faults are published by the respective MPI implementations.
2. Integrating Open MPI with the FTB.
3. Improving the fault tolerance and resiliency of Open MPI.

The rest of this section discusses the progress made in these three areas.

Standardized FTB Events for MPI Implementations With input from MPI users and developers the CIFTS team standardized fault events to be subscribed to and published by MPI libraries and runtime systems. The *FTB MPI Standardized Events document version 1.0* [46] was released in November 2010 at the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'2010) and describes a dozen fault events and their relevant attributes that are relevant to *all* MPI implementations. These fault events are categorized as (1) error events, such as failed, unreachable, or aborted processes or failed migration or checkpoint/restart operations; (2) warnings, such as transient communication errors; and (3) information events, such as notification of completed checkpoints or process migrations.

The MPICH, MVAPICH, and Open MPI groups have integrated FTB into their MPI implementations and are compliant with the FTB MPI Standardized Events document version 1.0.

Checkpoint/Restart in Open MPI Indiana University has been working closely with both the user and developer communities to improve the stability and performance of checkpoint/restart implementations in Open MPI, an implementation of the Message Passing Interface (MPI) standard. Our goal in this regard is to produce and maintain a high-quality transparent checkpoint/restart implementation in Open MPI that encourages both application developers and researchers to experiment with it in their domains. In addition to correctness and performance tuning we have demonstrated that Open MPI is able to checkpoint and restart a series of benchmark and real MPI applications, including the NAS parallel benchmarks, High-Performance Linpack (HPL), Parallel Ocean Program (POP), and the GROMACS and Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) molecular dynamics package. In an effort to support a variety of platforms and improve performance we are expanding our support for interconnects beyond Ethernet to include shared memory and high speed interconnects such as InfiniBand and Myrinet.

We have added support for a variety of interconnects including TCP/IP, shared memory, InfiniBand and Myrinet. A unique feature of our research is the ability to reconfigure interconnect pairings for improved performance on restart. This work was included as a part of the Open MPI v1.3 release series. We have also added support for checkpoint/restart-enabled transparent proactive process migration, and reactive automatic recovery. The proactive process migration feature will allow end users to move processes away from predicted failure and planned system outages. The reactive automatic recovery feature will provide end users with a transparent, automatic recovery mechanism when an unexpected process failure occurs. Alongside this work, we have improved the checkpoint stable storage mechanisms to support centralized and staged techniques. Staging checkpoint files to stable storage overlaps the writing of checkpoint files with application execution ultimately leading to a significant reduction in application performance overhead. As part of the staging technique, we have added caching and compression of checkpoint files. Caching checkpoint files improves automatic recovery time by referencing a local copy of a checkpoint when available. Compression often reduces the size of the checkpoint files and results in a reduction in the time to checkpoint and disk space required to do so.

Currently the Open MPI project's checkpoint/restart functionality depends upon the Berkeley Laboratory Checkpoint/Restart (BLCR) project. BLCR provides Open MPI with a system-level, transparent single process checkpoint/restart service. Over the course of this project we have collaborated with the BLCR project to stabilize existing interfaces, and experiment with new interfaces. One such interface collaboration is the *hook* interface provided to support checkpoint/restart-

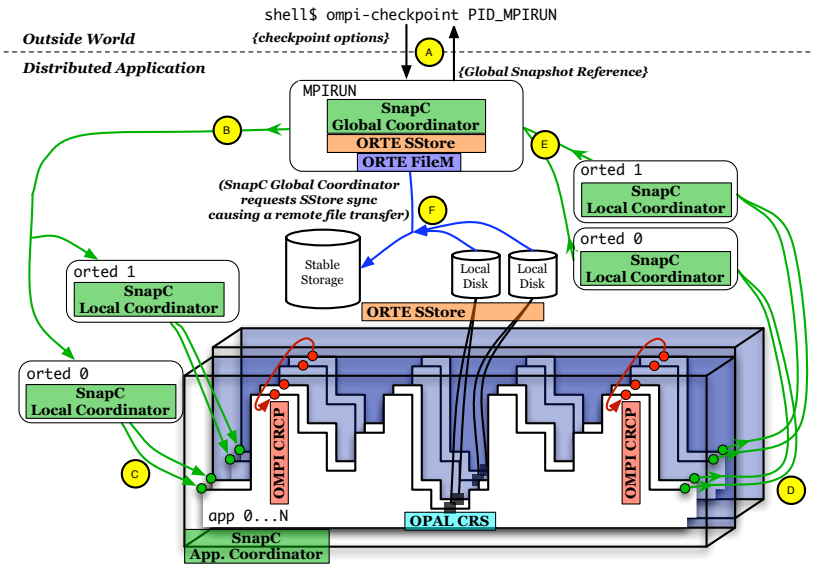

Figure 3: Illustration of Open MPI C/R frameworks participating in a distributed checkpoint of a running MPI application. 3D boxes represent nodes containing white application processes. Rounded boxes represent runtime support processes.

enabled parallel debugging. The collaborative partnership between the Open MPI and BLCR projects has allowed for peer evaluation of each other's software on different platforms and in different application domains than typically available to each individually. Through this collaboration we have helped investigate and solve a number of software bugs in both the Open MPI and BLCR projects resulting in a more stable checkpoint/restart experience for both single and parallel processing applications. Alongside this work, we have also improved the checkpoint stable storage mechanisms to support centralized and staged techniques. Staging checkpoint files to stable storage overlaps the writing of checkpoint files with application execution ultimately leading to a significant reduction in application performance overhead. As part of the staging technique, we have added caching and compression of checkpoint files. Caching checkpoint files improves automatic recovery time by referencing a local copy of a checkpoint when available. Compression often reduces the size of the checkpoint files and results in a reduction in the time to checkpoint and disk space required to do so.

The most time consuming component of the software development life-cycle is application debugging. This year we added support for checkpoint/restart-enabled parallel debugging in Open MPI that can dramatically shorten the debugging cycle. Software developers can save hours or days of time spent debugging by checkpointing and restarting the parallel debugging session at intermediate points in the debugging cycle. We also introduced a variety of checkpoint/restart application interfaces through the Open MPI Extensions interface. These interfaces provide applications with the opportunity to guide the checkpoint/restart related operations to best suit the application requirements. In addition to a checkpoint and a restart interface, we also expose interfaces to migrate processes within an MPI communicator, and receive notification of the progress of a checkpoint.

Open MPI Runtime Environment (ORTE) The underlying runtime support layer of Open MPI, known as Open MPI Runtime Environment (ORTE), has undergone considerable development by the Open MPI team members over the past year, primarily focusing on improving scalability and reliability. We participated in some of the ORTE code rework and re-factoring through the development of the Runtime Services Layer (RSL) interface. The RSL is a software engineering approach to isolate the rest of the MPI implementation from the internals of ORTE, and ultimately, if adopted, will allow Open MPI to utilize different runtime environments. Although the RSL has not yet been incorporated into the mainline Open MPI code, it has greatly influenced the development and simplification of ORTE itself. We continue to work with other Open MPI developers to improve ORTE to not only be more scalable, but also more robust and fault tolerant on the system targeted by the Coordinated Infrastructure for Fault Tolerant Systems (CIFTS) project. We added support for process fault recovery into the Error Management (ErrMgr) framework. The fault recovery extensions allowed us to add support for checkpoint/restart-enabled, transparent proactive process migration, and reactive automatic recovery. The new framework also supports MPI applications that choose to run-through a process failure by stabilizing the runtime environment and continuing execution. This stabilization recovery feature supports ongoing research into fault tolerant MPI semantics, currently under consideration by the Fault Tolerance Working Group in the MPI Forum.

The stabilization recovery feature directly supports our efforts to support task farm and manager/worker styles of parallelism in Open MPI. At Supercomputing 2009, we presented a modified version of the POV-Ray parallel application that was able to use the stabilization recovery feature and the CIFTS Fault Tolerance Backplane (FTB) to run-through a process failure and correctly finish execution. We refined MPI and CIFTS interfaces, and improve the robustness of ORTE to support fault tolerant task farm and manager/worker styles of parallelism. One of the other demos at Supercomputing 2010 involved proactive migration of MPI processes due to predicted failures. Impending failures were predicted by monitoring FTB for fault information. We also demonstrated reactive fault tolerance in Open MPI where an MPI job sustained execution despite failures owing to the new run-through stabilization features in the runtime. In line with this work, we have developed a test suite for the task farm scenario that will be added to the nightly Open MPI regression tests. This test suite checks for the ability of the MPI implementation to allow a specialized MPI program to continue to run after a node failure.

Integrating the FTB with Open MPI The FTB provides a common shared infrastructure for system software components to exchange fault information and coordinate responses through a uniform interface. High Performance Computing (HPC) system software, such as Open MPI, use the FTB to interact with other FTB-enabled components in the system to add to the overall resiliency of the system.

FTB support in Open MPI uses the FTB client API (version 0.5) specification to exchange fault-related information with the FTB. Oftentimes, the MPI implementation is amongst the first to detect faults in a running parallel application. Upon fault detection (or suspicion), Open MPI can relay the information about the fault to other components over the FTB and/or act on the fault locally. Additionally, Open MPI may listen to events from other FTB-enabled components and handle these events depending on the type of the fault or the action requested. For example, Open MPI can initiate a coordinated checkpoint of the running parallel processes on receiving the

corresponding event from a FTB-enabled job scheduler.

The FTB support in Open MPI is implemented as a component of the Notifier framework in ORTE. The Notifier framework exports information, warnings and errors related to Open MPI-detected problems to one or more Notifier components which are selected at runtime. Leveraging the infrastructure provided by the Open Portable Access Layer (OPAL) SOS interface, Open MPI can control the way that these events are reported. Fault events can be reported to multiple Notifier components, including the FTB, and then acted upon appropriately by calling the necessary internal library routines. Further, OPAL SOS provides Open MPI the opportunity to filter, aggregate or coalesce events according to severity and others parameters. The FTB Notifier component is available in the Open MPI trunk (as of r20655). It is currently deemed experimental and full support for FTB is expected to be included in the next feature release of Open MPI in the v1.5 series.

Multi-level content-addressable checkpoint file system We developed a multilevel, content-addressable checkpoint file system which achieves in-flight checkpoint data reduction across all compute nodes through compression and elimination of duplicate blocks over a series of checkpoints. Through a detailed analysis of checkpoint dumps, we assessed the benefits of data reduction for scientific applications that are representative of production workloads. The reduction in checkpoint commit latencies was shown to minimize the total execution time (including the dump, restart and rework times) of an application. We further implemented an analytical model for multilevel checkpoint/restart I/O to develop insights into co-design of exascale storage infrastructure.

4.4 Fault Tolerance and the MPI Standard

Indiana University is currently an active participant in the recently reconvened MPI Forum. We assisted in the standardization process for MPI 1.3 (July 2008) and MPI 2.1 (September 2008). We are currently assisting with the current MPI 2.2 standardization effort. In addition we are participating in a number of MPI 3.0 working groups, each charged with investigating interface and wording adjustments to better support current and next generation HPC applications.

The MPI 3.0 Fault Tolerance Working Group is charged with investigating MPI standard adjustments that will enable applications to explore a wide range of fault tolerance techniques. A number of proposals have been brought forward for discussion including two from Indiana University.

The first proposal presented for discussion is the addition of a checkpoint/restart quiescence API. This API gives the application control over when and how checkpoints occur from within the application. This level of control will allow applications to better use a wider variety of checkpoint/restart services available to them at runtime, including user and system level services. A prototype implementation is available as part of the Open MPI Extensions interface in the current trunk, and will be incorporated into the v1.5 release series. The proposal was presented to the Fault Tolerance Working Group in early 2009, and the resulting feedback was integrated into the proposal. The working group highlighted some portability issues with the interface that are currently under investigation by the concerned parties.

The second proposal is improvements to the process creation and management interface originally presented in MPI 2.0. This proposal strengthens the wording in the standard regarding MPI implementation expectations in the presence of failures. In addition, this proposal presents

a nonblocking process creation and management interface. The nonblocking interface will allow an application to overlap process creation and connection establishment with computation. These two parts of the proposal will allow applications the ability to prepare for and react to the failure of a process depending on the applications requirements in a standard way across MPI implementations.

4.5 FTB support for system components

Error logger The *syslog* protocol has been the de-facto industry standard for logging event notification messages generated by programs. *syslog* clients are bundled with almost every operating system distribution. Nearly all monitoring services running on networked machines have plugins that interface with *syslog*. Given its prevalent usage, several data-mining and analytical tools have been developed for recognition, aggregation, and correlation of *syslog* events. It provides a logging infrastructure commonly utilized by programs and services across the entire software stack. Thus, *syslog* provides important information about a system by logging all the events that are configured to be monitored. To leverage the pervasive presence of *syslog*, we developed a FTB-*syslog* software that relays event notifications between *syslog* and FTB. The software publishes *syslog* messages of interest so that other FTB-aware components can subscribe to them and take relevant actions. Software services that are agnostic of FTB, thus, indirectly act as sources of failure event notifications to help in making holistic fault tolerance decisions. Further, active decisions made by FTB components are also logged to *syslog* for provenance. We have developed an application to catch every new *syslog* event based on the *syslog* configuration and publish it to the FTB by using the FTB and Python bindings. Since the log formats and configuration policies of *syslog* vary on different systems, it is required to make the application generic to coordinate with the target systems. The FTB and Python binding used for this purpose make it easy to adapt to a variety of the system logs. To filter events of interest from the *syslog* stream, we plan to investigate developing a continuous query language (CQL) that can help in specifying precise constraints and range-queries on the generated events. This would help in finding surprising patterns in the incoming data stream, establishing correlation between distinct temporal events and reducing the overall noise in events published to FTB.

Resource Manager Simple Linux Utility for Resource Management (SLURM) is a popular, open-source resource manager and job scheduler developed by Lawrence Livermore National Laboratory. SLURM is actively managing resources on several of the TOP500 supercomputers including the Tianhe-1A supercomputer at NUDT and the Tera-100 at CEA. SLURM is designed to support heterogeneity of resources, be portable and extensible through its use of a sophisticated plugin system and tolerate system failures including node failures executing its own control functions. It continues to be actively developed to support new architectures, interconnects, authentication mechanisms and job scheduling policies.

We extended SLURM by adding a new notifier plugin to report events to the FTB. Notifications related to monitoring of resources, scheduling of jobs and failure events internal to SLURM are supported. The SLURM controller daemon (*slurmctld*) publishes these events to FTB through its various hooks using the notifier plugin. FTB-aware components interested in these events can, thus, track resource changes, job status and SLURM failures. SLURM can consequently be con-

trolled externally through its command-line tools, a library interface or using an external scripting language. FTB integration of SLURM provided us insights into commonly occurring resource and job failures, which led to the initial efforts in FTB fault event standardization pertaining to RM/JS. In addition to the supported failure and status events, we plan to add support for FTB event notifications related to resource usage and accounting, QoS parameters and overall job statistics. This information is invaluable in improving the cost-effectiveness of the machine and taking intelligent decisions about management of the available resources. With the planned addition of dynamically resizable jobs and resource pools using hot-spares in SLURM, we intend to extend it to listen for event notifications from other FTB-aware components to offer more tightly-coupled fault tolerance.

5 Outreach and Education Activities

The CIFTS team has been involved in various outreach efforts during the CIFTS project. Following is a comprehensive list of all efforts:

1. The team has designed publicly accessible web resources for dissemination of CIFTS-related research: (1) CIFTS web page [21] provides overall status and progress of the project, (2) the CIFTS wiki [22] serves as a repository for all detailed documents for all aspects of CIFTS research, including weekly/other meetings minutes, (3) the CIFTS SVN [18] serves as a repository for code development, presentations, papers, meetings, and supporting documents, and (4) the CIFTS TRAC [20] is used for tracking bug fixes, future features, and enhancements. Most of the information related to the project is readily available to provide transparency and foster collaborations.

   In addition, the software that has been FTB-enabled has individual project websites that have also been used for information sharing.

2. The CIFTS team has over *100 outreach materials* in various forms (see bibliography for detailed CIFTS-related articles and talks). In particular, we have published approximately 30 publications, presented 40 talks and 5 posters, and conducted Birds-of-a-Feather sessions and round-table discussion sessions every year for the past four years at the IEEE/ACM Supercomputing conference, and we have given more than 25 demonstrations of our research at various venues. The FTB design paper [36] has been cited two dozen times and by several independent sources, in less than two years. The community-targeted Birds-of-a-Feather sessions held at the IEEE/ACM Supercomputing conference [24–27] have been popular, with more than 50 attendees each year.

3. The CIFTS team has encouraged and fostered extensive collaboration between their internal team as well as with external teams at other institutes. Approximately half a dozen students belonging to CIFTS institutes and external institutes such as the Illinois Institute of Technology and North Carolina State University have collaborated as summer interns or research assistants on various aspects of the CIFTS framework.

4. Members of the CIFTS team have collaborated with several vendors and other research groups. Prominent among them are our collaborations with the SLURM, TORQUE, FEDORA, and DEBIAN teams, which have resulted in integrating support for FTB-enabled software such as BLCR with their software. Equally noteworthy are our collaborations with IBM and CRAY, which have resulted in outlining clearer goals for end-to-end fault management in future systems as well as defining the new FTB API 1.0 specification.

5.1 Students Supported

The award supported four graduate students and three postdoctoral researchers at Indiana University.

5.2 Publications

[1] Rinku Gupta, Pete Beckman, Hoony Park, Ewing Lusk, Paul Hargrove, Al Geist, Dhabaleswar K. Panda, Andrew Lumsdaine, and Jack Dongarra. CIFTS: A Coordinated infrastructure for Fault-Tolerant Systems. International Conference on Parallel Processing (ICPP), September 2009.

[2] Joshua Hursey, Chris January, Mark O'Connor, Paul H. Hargrove, David Lecomber, Jeffrey M. Squyres, and Andrew Lumsdaine. Checkpoint/restart-enabled parallel debugging. *Proceedings of the European MPI Users Group Conference (EuroMPI)*, September 2010.

[3] Joshua Hursey and Andrew Lumsdaine. A composable runtime recovery policy framework supporting resilient HPC applications. Under submission, 2010.

[4] Joshua Hursey, Timothy I. Mattox, and Andrew Lumsdaine. Interconnect agnostic checkpoint/restart in Open MPI. In *HPDC '09: Proceedings of the 18th ACM international symposium on High Performance Distributed Computing*, pages 49–58, New York, NY, USA, 2009. ACM.

[5] Joshua Hursey, Jeffrey M. Squyres, Timothy I. Mattox, and Andrew Lumsdaine. The design and implementation of checkpoint/restart process fault tolerance for Open MPI. In *Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1 – 8. IEEE Computer Society, March 2007.

[6] Abhishek Kulkarni, Adam Manzanares, Latchesar Ionkov, Michael Lang, and Andrew Lumsdaine. The Design and Implementation of a Multi-level Content-Addressable Checkpoint File System. In *Proceedings of the 19th International Conference on High Performance Computing (HiPC 2012)*, December 2012.

5.3 Talks

We gave a talk at the IEEE Supercomputing 2006 conference about our work on fault tolerance [37]. We presented our work on fault tolerance in Open MPI to Sun [39] and to the Innovative

Computing Laboratory at the University of Tennessee [38]. We gave three talks at the Supercomputing 2007 conference, one each at the Cisco and Indiana University Booths [42], and one during the CIFTS Birds-of-a-Feather meeting. We gave two talks at the Indiana University Booth at the Supercomputing 2008 conference about our work on CIFTS [40, 43]. We gave three talks at the IEEE Supercomputing 2009 conference, one at the Cisco Booth, one at the Argonne National Laboratory Booth, and a mini-workshop at the Indiana Booth [11–13]. At Supercomputing 2010, we organized a demo at the Indiana University booth and a talk at the Argonne National Laboratory booth [41] about the integration of the fault tolerance backplane with Open MPI.

[7] Joshua Hursey. Fault Tolerance in Open MPI. Presentation in the Indiana University booth at the ACM/IEEE SC06 Conference, Tampa, FL, November 2006.

[8] Joshua Hursey. Process Fault Tolerance in Open MPI. Presentation to Innovative Computing Laboratory (ICL) Friday Lunch Speaker Series, University of Tennessee, Knoxville, Feburary 2007.

[9] Joshua Hursey. Checkpoint/Restart Support in Open MPI. Presentation to Sun Microsystems, Inc. Tech Talk Series, May 2008.

[10] Joshua Hursey. Fault Tolerance in High Performance Computing: MPI and Checkpoint/Restart. Presentation in the Indiana University booth at the ACM/IEEE SC08 Conference, Austin, Texas, November 2008.

[11] Joshua Hursey. A Transparent Process Migration Framework for Open MPI. Presentation in the Cisco booth at the ACM/IEEE SC09 Conference, Portland, Oregon, November 2009.

[12] Joshua Hursey, Jeffrey M. Squyres, Abhishek Kulkarni, and Andrew Lumsdaine. Open MPI Tutorial. Presentation in the Indiana University booth at the ACM/IEEE SC09 Conference, Portland, Oregon, November 2009.

[13] Abhishek Kulkarni. Process Resilience in Open MPI using the CIFTS Fault Tolerance Backplane: A POV-Ray Demonstration. Presentation in the Argonne National Laboratory booth at the ACM/IEEE SC09 Conference, Portland, Oregon, November 2009.

[14] Abhishek Kulkarni. Fault Tolerance in Open MPI using the FTB. Presentation at the Argonne National Laboratory booth at the ACM/IEEE SC10 Conference, New Orleans, Louisiana, November 2010.

[15] Timothy I. Mattox. MPI Is Dead? Long Live MPI! Evolving MPI for the Next Generation of Supercomputing. Presentations in the Cisco and Indiana University booths at the ACM/IEEE SC07 Conference, Reno, Nevada, November 2007.

[16] Timothy I. Mattox. Research at Indiana University for Reliable Petascale Performance. Presentation in the Indiana University booth at the ACM/IEEE SC08 Conference, Austin, Texas, November 2008.

5.4 Technical Reports

[17] Joshua Hursey, Jeffrey M. Squyres, and Andrew Lumsdaine. A checkpoint and restart service specification for Open MPI. Technical Report TR635, Indiana University, Bloomington, Indiana, USA, July 2006.

5.5 Web Sites

The Open MPI project maintains several websites hosted by Indiana University, primarily a main public website, and a developer wiki and bug tracker:

- `http://www.open-mpi.org/`
- `https://svn.open-mpi.org/trac/ompi/wiki`

The CIFTS project overall maintains two websites hosted by Argonne National Lab, similarly split between a main public website and a developer wiki:

- `http://www.mcs.anl.gov/research/cifts/`
- `http://wiki.mcs.anl.gov/cifts/index.php`

In addition to the Open MPI and CIFTS project websites, we also maintain a Fault Tolerance research webpage. This webpage provides documentation and detailed examples for all of the fault tolerance related work going on at Indiana University.

- `http://osl.iu.edu/research/ft/`
- `http://svn.osl.iu.edu/trac/ftb/wiki/syslog`

6 Deliveries and Contributions

6.1 Deliveries

In this project, we undertook the following software deliveries:

- Open MPI with coordinated and uncoordinated fault tolerance
- Fault-aware Open MPI with FTB support
- FTB system components (error logger and resource manager)
- A multi-level content-addressable checkpoint file system
- Distributed testing infrastructure to aid FTB development and deployment

References

[18] SVN for Coordinated Infrastructure for Fault Tolerant Systems. https://svn.mcs.anl.gov/repos/cifts.

[19] Top 500 Supercomputer Sites. http://www.top500.org/.

[20] TRAC for Coordinated Infrastructure for Fault Tolerant Systems. http://trac.mcs.anl.gov/projects/cifts.

[21] WEBPAGE for Coordinated Infrastructure for Fault Tolerant Systems. http://www.mcs.anl.gov/research/cifts/.

[22] WIKI for Coordinated Infrastructure for Fault Tolerant Systems. http://wiki.mcs.anl.gov/cifts/index.php.

[23] Argonne National Laboratory. MPICH2 . http://www.mcs.anl.gov/research/projects/mpich2/.

[24] P. Beckman, D. Bernholdt, D. K. Panda, P. Hargrove, A. Bouteiller, and A. Kulkarni. CIFTS: Coordinated Fault Tolerance for High Performance Computing. BOF on CIFTS, in conjunction with the ACM/IEEE International Conference for High Performance Computing (HPC), Networking, Storage and Analysis (SC,10), November 2010.

[25] P. Beckman, D. Bernholdt, D. K. Panda, P. Hargrove, A. Bouteiller, and A. Lumsdaine. CIFTS: Coordinated Fault Tolerance for High Performance Computing. BOF on CIFTS, in conjunction with the ACM/IEEE International Conference for High Performance Computing (HPC), Networking, Storage and Analysis (SC) (SC '07), November 2007.

[26] P. Beckman, D. Bernholdt, D. K. Panda, P. Hargrove, A. Bouteiller, and A. Lumsdaine. CIFTS: Coordinated Fault Tolerance for High Performance Computing. BOF on CIFTS, in conjunction with the ACM/IEEE International Conference for High Performance Computing (HPC), Networking, Storage and Analysis (SC'08), November 2008.

[27] P. Beckman, D. Bernholdt, D. K. Panda, P. Hargrove, A. Bouteiller, and A. Lumsdaine. CIFTS: Coordinated Fault Tolerance for High Performance Computing. BOF on CIFTS, in conjunction with the ACM/IEEE International Conference for High Performance Computing (HPC), Networking, Storage and Analysis (SC'09), November 2009.

[28] R. Gupta. CIFTS: Coordinated Infrastructure for Fault Tolerant Systems. Talk in Argonne National Laboratory booth at the IEEE/ACM International Conference for High-Performance Computing, Networking, Storage and Analysis (SC), November 2007.

[29] R. Gupta. Introduction to CIFTS. Talk at the CCA Forum meeting, July 2007.

[30] R. Gupta. CIFTS: Coordinated Infrastructure for Fault Tolerant Systems. Talk at the Argonne National booth at the IEEE/ACM International Conference for High-Performance Computing, Networking, Storage and Analysis (SC), November 2008.

[31] R. Gupta. CIFTS: Coordinated Infrastructure for Fault Tolerant Systems. Talk at the Workshop on Fault Tolerance and Resiliency, In conjunction with Los Alamos Computer Science Symposium (LACSS), October 2008.

[32] R. Gupta. CIFTS: Coordinated Infrastructure for Fault Tolerant Systems. Invited Talk at Fermi National Accelerator Laboratory (Fermilab), May 2009.

[33] R. Gupta. CIFTS: Coordinated Infrastructure for Fault Tolerant Systems. Talk at the Argonne Leadership Computing Facility (ALCF), May 2009.

[34] R. Gupta. CIFTS: Coordinated Infrastructure for Fault Tolerant Systems. Invited talk at University of Chicago, April 2009.

[35] R. Gupta. CIFTS: Coordinated Infrastructure for Fault Tolerant Systems. Invited Talk at SIAM Conference on Parallel Processing for Scientific Computing, February 2010.

[36] Rinku Gupta, Pete Beckman, Byung-Hoon Park, Ewing Lusk, Paul Hargrove, Al Geist, Dhabaleswar Panda, Andrew Lumsdaine, and Jack Dongarra. CIFTS: A Coordinated Infrastructure for Fault-Tolerant Systems. *International Conference on Parallel Processing (ICPP)*, pages 237–245, 2009.

[37] Joshua Hursey. Fault Tolerance in Open MPI. Presentations in the Indiana University booth at the ACM/IEEE SC06 Conference, Tampa, FL, November 2006.

[38] Joshua Hursey. Process Fault Tolerance in Open MPI. Presentation to Innovative Computing Laboratory (ICL) Friday Lunch Speaker Series, University of Tennessee, Knoxville, Feburary 2007.

[39] Joshua Hursey. Checkpoint/Restart Support in Open MPI. Presentation to Sun Microsystem, Inc. Tech Talk Series, May 2008.

[40] Joshua Hursey. Fault Tolerance in High Performance Computing: MPI and Checkpoint/Restart. Presentation in the Indiana University booth at the ACM/IEEE SC08 Conference, Austin, Texas, November 2008.

[41] Abhishek Kulkarni. Fault Tolerance in Open MPI using the FTB. Presentation at the Argonne National Laboratory booth at the ACM/IEEE SC10 Conference, New Orleans, Louisiana, November 2010.

[42] Timothy I. Mattox. MPI Is Dead? Long Live MPI! Evolving MPI for the Next Generation of Supercomputing. Presentations in the Cisco and Indiana University booths at the ACM/IEEE SC07 Conference, Reno, Nevada, November 2007.

[43] Timothy I. Mattox. Research at Indiana University for Reliable Petascale Performance. Presentation in the Indiana University booth at the ACM/IEEE SC08 Conference, Austin, Texas, November 2008.

[44] Network-Based Computing Laboratory. MVAPICH/MVAPICH2: MPI-1/MPI-2 for InfiniBand and iWARP with OpenFabrics. `http://mvapich.cse.ohio-state.edu`.

[45] Open MPI Group. Open MPI: Open Source High Performance Computing. http://www.open-mpi.org.

[46] The CIFTS Team. FTB MPI standardized events version 1.0: http://www.mcs.anl.gov/research/cifts/. http://www.mcs.anl.gov/research/cifts/, November 2010.