Cover Page

Title of Project:

Coordinated Fault Tolerance for High-Performance Computing Final Project Report
FWP #57613

Office of Science Announcement Title/#:

Argonne National Laboratory

Principal Investigator(s):

Peter Beckman, Director, Exascale Technology & Computing Institute (ETCi)
Argonne National Laboratory
Exascale Technology & Computing Institute
9700 So. Cass Avenue - Bld.240
Lemont, IL 60439
(T) 630-252-9020 (Fax) 630-252-2828
beckman@mcs.anl.gov

Official signing for Laboratory:

Rick L. Stevens, Associate Laboratory Director
Argonne National Laboratory
Computing, Environment, & Life Sciences (CELS)
9700 So. Cass Avenue - Bld.240
Lemont, IL 60439
(T) 630-252-3378, (Fax) 630-252-6333
stevens@anl.gov

Duration of Entire Project Period:

7/01/2006 to 06/30/2011

Use of human subjects in proposed project:	No
Use of vertebrate animals in proposed project:	No

Signature of PI, Date of Signature:

Peter Beckman

July 28, 2011

Signature of Official, Date of Signature:

[illegible]

July 28, 2011

Table of Contents

1 Research Summary

1.1 Motivation

The main purpose of the Center for the Improvement of Fault Tolerance in Systems 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-end computing (HEC) 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 middleware or 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 middleware/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.

From a broad perspective, our work to meet our goal of end-to-end fault tolerance has focused on two areas: (1) improving fault tolerance in various software currently available and widely used throughout the HEC domain and (2) using fault information exchange and coordination to achieve holistic, systemwide fault tolerance and understanding how to design and implement interfaces for integrating fault tolerance features for multiple layers of the software stack—from the application, math libraries, and programming language runtime to other common system software such as jobs schedulers, resource managers, and monitoring tools.

1.2 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 HEC 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 HEC 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 HEC system software lacks today that would inhibit such software from taking advantage of systemwide global fault information?

- What are the practical limitations of a systemwide 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 on 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 parallel 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.

  Targeting 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-end 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.

1.3 Accomplishment Highlights

Following are some of the highlights of the CIFTS project:

1. 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.

2. The three most widely deployed implementations of the Message Passing Interface (MPI) standard, MPICH2, MVAPICH2, and Open MPI, have been enhanced to integrated with the FTB, and support a common set of fault-related events. Additionally, all three libraries have significantly improved their support for checkpoint/restart.

3. With support from the CIFTS project, the Berkeley Lab Checkpoint/Restart (BLCR) package has been significantly enhanced, integrated with the FTB, SLURM, and TORQUE, as well as all three MPI implementations (as mentioned above). BLCR has also been ported to several new platforms and added to several Linux distributions.

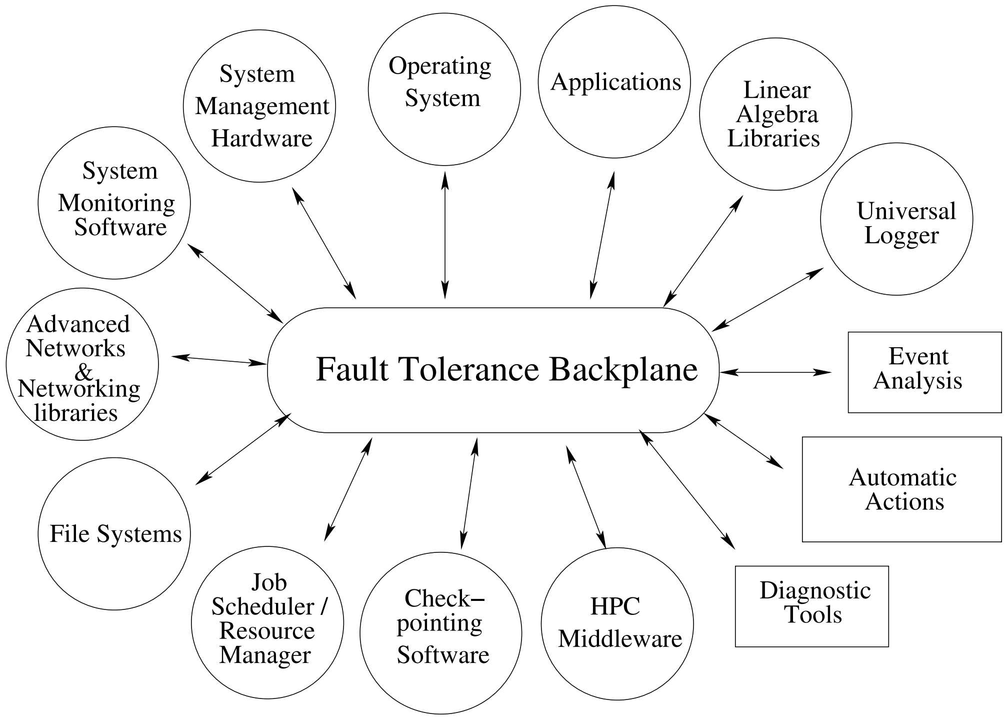

Figure 1: CIFTS Framework

4. The FT-Linear Algebra library has been FTB-enabled as a prototype, and the development of a fault tolerance version of the ScaLAPACK is underway which uses algorithm-based fault tolerance to provide applications an effective alternative to traditional checkpoint/restart.

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 make 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.

2 Technical Approach and Research Accomplishments

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.

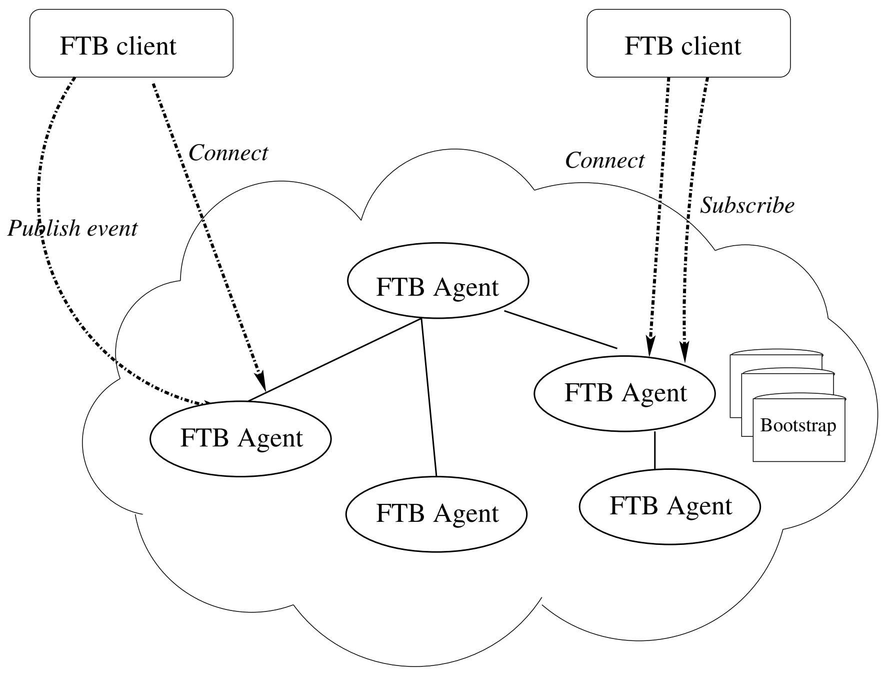

Figure 2: The FTB Architecture

2.1 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 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 [4].

2.2 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 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.

Details about the design of the FTB implementation can be found in [38].

Details on CIFTS and the FTB software implementation have been a focus of several talks and presentations [30–37] given during this project.

2.3 Other FTB API Implementations

To reach out to other communities, in addition to the CIFTS FTB software, the CIFTS team studied and ported the FTB API version 0.5 to other existing commodity communication middleware. The CIFTS team noted that significant effort had been made to standardize and implement communication APIs in areas such as *service availability* and *enterprise message exchange* and that highly reliable commodity implementations were already available. The CIFTS team found that FTB implemented based on this middleware provides not only reliable event exchange capability but also seamless integration with the applications that utilize the communication middleware, thereby spreading the CIFTS concept to the community. The CIFTS team particularly chose the following two widely accepted communication specifications and their implementations.

- SAF Application Interface Specification (AIS).
- Advanced Message Queueing Protocol (AMQP).

The Service Availability Forum (SAF) [114] was established to foster creation of highly available network infrastructure products, systems and services. An implementation of FTB was made with the Event Distribution Service (EDSv) of SAF's Application Interface Specification. The EDSv consists of an Event Distribution Server (EDS) and Event Distribution Agent (EDA). Event messages are published and received through local EDAs that connect to the central EDS. A message routed from the EDS is delivered only when the attribute attached with the message passes through a filter set by the subscriber. Utilizing the attribute filters of EDSv, the CIFTS team made the FTB APIs available for applications that run on systems where a EDS is in operation. For an implementation of the Application Interface Specification, openSAF [113], which is available under the LGPLv2.1 license, was selected.

The Advanced Message Queuing Protocol (AMQP) [131] is an open standard application layer protocol for message-oriented middleware. Although AMQP originated in the financial sector, it has gained applicability to a broad range of middleware domains. The heart of the AMQP service framework is an *exchange*, which receives messages from publishers and routes them to message queues depending on predefined rules attached to the queues. A message queue is created dynamically by a subscriber. The CIFTS team implemented the FTB API by defining publish/subscribe as bindings between the exchange and message queues, especially by matching routing keys using the *topic exchange* method. Two open-source implementations of AMQP—openAMQ [108], which is licensed under GPLv3 and managed by iMatix, and Qpid [112], which is licensed under the Apache license version 2 and managed by Apache, RedHat, and various contributors—were selected to implement the FTB API.

Additional information on these designs can be found on the CIFTS wiki [5].

2.4 Improving Fault Tolerance in Software Components

This section focuses on software that currently is part of the CIFTS infrastructure and integrates with the Fault Tolerant Backplane. In addition to the integration details, we also briefly discuss the research and design advancements made to this software in order to enable it to work in coordinated fault management environments.

2.4.1 The Message Passing Interface Libraries

The Message Passing Interface (MPI) is one of the most important programming models in high-performance computing. MPICH2, [116], Open MPI [127], and MVAPICH2 [126] are three of the most popular MPI implementations that heavily dominate the high-performance computing space [115]. The three teams in the CIFTS project have focused on three areas:

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

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* [100] 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.

Fault Tolerance in MPICH2 The MPICH2 team, as of Feb 2010, provides fault tolerance support for MPICH2. This support is available in the MPICH2 software since version 1.3 [116]. Fault tolerance in MPICH2 is provided through a checkpoint/restart mechanism which is based on the Berkeley Lab Checkpoint/Restart software. We have taken advantage of features in BLCR that allows us to take a checkpoint of every process on a node all at once. This allows preservation of shared-memory channels between the processes. Checkpointing is integrated with MPICH's Hydra process manager which publishes FTB events reporting the progress of the checkpoint operation.

The MPICH2 team has also worked with the Fault-Tolerance Working Group of the MPI Forum to develop a fault-tolerance model and API for MPI. This work has resulted in a proposal being presented to the MPI Forum [26], as well as two posters [21,54] and a technical report [22].

FTB Integration in MPICH2 As of MPICH2 version 1.3.1, MPICH2 supports all fault events listed in FTB MPI Standardized Events document version 1.0. In MPICH2, fault events are either published by the MPI library itself or by the Hydra process manager.

We have presented several presentations [7–10] and demonstrations [20,23] showcasing the capabilities of FTB-enabled MPICH2 with other FTB-enabled software. In particular, we showcased [23] detection of failed communication processes on a cluster–fault information about which was communicated by MPICH2 through FTB. This information was obtained by the FTB-enabled COBALT job scheduler, which removed the failed network links (and nodes connected to them) from the resource pool.

Fault Tolerance in MVAPICH/MVAPICH2 (MPI on InfiniBand) Several fault tolerance features have been added to the MVAPICH and MVAPICH2 software, under the CIFTS initiative, to make it highly available and resilient for deployment on leadership class machines and to increase its usefulness in coordinated fault management environments. Following is a description of some of the more important fault tolerant features, that are now available in the MVAPICH/MVAPICH2 software.

End-to-End Reliable Data Transmission in MVAPICH: The MVAPICH team has designed and incorporated an end-to-end reliable data transmission protocol in the MPI layer. This design is able to detect error using CRC encoding and retransmit erroneous data upon a corrupted packet. This feature is available in the current MVAPICH distribution.

Automatic Path Migration (APM) Support in MVAPICH2 for InfiniBand Networks: InfiniBand provides a hardware mechanism, Automatic Path Migration (APM), which allows user transparent detection and recovery from network fault(s). We have designed a set of modules [6], namely: (a) an Alternate path specification module, (b) a Path loading request module and (c) a Path migration module, which work together to provide network fault tolerance for user-level applications. This support has been added to MVAPICH and MVAPICH2 software and is available in their respective distributions.

Reactive Process-Level Fault-Tolerance in MVAPICH2: MVAPICH2 has built-in support for coordinated process-level checkpoint/restart, a most widely deployed reactive fault-tolerance strategy. We have designed a complete checkpoint/restart framework with a set of coordination protocols [95] to guarantee the channel consistency of InfiniBand and intra-node shared memory channels during a checkpoint cycle. The BLCR library is used to create a system-level snapshot of each MPI process. Basic Checkpoint/Restart is well known for its IO bottleneck while saving process snapshots to shared file system. We have proposed a set of optimizations to tackle this constraint. A group-based checkpointing strategy is designed [96] that schedules the parallel processes to take checkpoint in smaller groups to reduce IO contentions during checkpoint writing. As multi-core processors are getting common for HPC systems, we worked out a novel I/O aggregation mechanism [71] to speedup checkpoint operations by reducing the concurrent IO overhead. Further enhancements with dynamic buffering have been carried out in [70] that use a small amount of buffer pool to accelerate checkpoint operations further. These schemes have been evaluated in conjunction with a new staging I/O framework and modern Solid State Disks (SSD) to provide a high-performance Checkpoint-Restart solution [72].

Proactive Process-Level Fault-Tolerance in MVAPICH2: In addition to the reactive Checkpoint/Restart strategy, we have implemented a proactive fault-tolerance scheme in MVAPICH2 to migrate processes on a failing node to a healthy spare node upon a failure prediction so as to bypass the IO bottleneck of a cluster-wide checkpoint. We have proposed a file-based migration design [102] that creates checkpoint files of the processes on the migration source node and move the files via different transports to the target node to restart the processes. The team has made significant enhancements in data transmission at [101] to directly pump process image through a RDMA data pipeline from migration source node to the target node without any file system IO overhead.

FTB Integration in MVAPICH/MVAPICH2 Support for FTB has been present in the MVAPICH2 software since version 1.4, which was publicly released in Nov. 2009. The current MVAPICH2 1.7 software supports the FTB API version 0.5 and is compatible with the FTB version 0.6 software. The MVAPICH2 software connects to the FTB backplane to subscribe and publish fault-related information. The current version of MVAPICH2 software supports all fault events described in the FTB MPI Standardized Events document version 1.0. MVAPICH2 subscribes to different kinds of FTB faults. Upon receipt of information about impending faults, MVAPICH2 can intelligently choose to take preventive proactive actions, such as systemwide full checkpoint to guard the current computation progress, or can perform a process-migration to evacuate the failure-prone node.

These capabilities of FTB-enabled MVAPICH have been demonstrated at several international conferences [68,69]. In particular, we demonstrated [69] how MVAPICH can effectively migrate the TACHYON [130] application process when it obtains fault information, published by the FTB-enabled InfiniBand monitoring networking library [66], through the FTB infrastructure.

The research results have been implemented in the MVAPICH/MVAPICH2 software and published [6, 38, 60, 70–72, 95, 96, 101, 102] and presented [7–10, 73–89] at several venues.

Fault Tolerance in Open MPI

The Open MPI team has added several new fault tolerance features to Open MPI. These enhancements include improvements and optimizations to the checkpoint/restart framework, such as reduced checkpoint times; improved extensions and mechanisms to the Open MPI runtime to support better fault recovery techniques; and integration of Open MPI with the FTB to exchange fault information.

Checkpoint/restart framework and service in Open MPI:

We have added support for a variety of interconnects including TCP/IP, shared memory, InfiniBand, and Myrinet. A unique feature of this research is the ability to reconfigure interconnect pairings for improved performance on restart [51,52,56]. Support has been added for checkpoint/restart-enabled transparent proactive process migration, and reactive automatic recovery [61, 64]. The proactive process migration feature allows end-users to move processes away from predicted failure locations and planned system outages [53]. The reactive automatic recovery feature provides end-users with a transparent, automatic recovery mechanism when an unexpected process failure occurs [58, 59]. 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 team has also added support for checkpoint/restart-enabled parallel debugging in Open MPI that can dramatically shorten the debugging cycle [55]. 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 have 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, interfaces to migrate processes within an MPI communicator and receive notification of the progress of a checkpoint has also been exposed. Currently the Open MPI project's checkpoint/restart functionality depends primarily on the BLCR project. BLCR provides Open MPI with a system-level, transparent, single-process checkpoint/restart service. Collaboration in this project stabilized existing interfaces and enabled experiments with new interfaces. One such interface collaboration is the *hook* interface provided to support checkpoint/restart-enabled parallel debugging.

Open Runtime Environment (ORTE) in Open MPI: The underlying runtime support layer of Open MPI, known as ORTE, has undergone considerable development by the Open MPI team members during the course of the CIFTS project, focusing primarily on improving scalability and reliability. We have participated in some of the ORTE code rework and refactoring through the development of the Runtime Services Layer (RSL) interface. The RSL interface proved to be invaluable in highlighting interface issues in ORTE that have since been addressed. We have added support for process fault recovery into the Error Management (ErrMgr) framework [56]. 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 [50].

Fault Tolerance Practice in Open MPI and the MPI Standard:

The Open MPI team at Indiana University is an active participant in the recently reconvened MPI Forum [49,52,57,63]. We assisted in the standardization process for MPI 1.3 (July 2008), MPI 2.1 (September 2008), and MPI 2.2 (September 2009). We are currently assisting with the current MPI 3.0 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.

FTB Integration in Open MPI FTB support in Open MPI supports the FTB API version 0.5 specification to exchange fault-related information with the FTB. Often, the MPI implementation is among 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 or act on the faults 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 OPAL (Open Portable Access Layer) SOS interface, Open MPI can control the way in which 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.

Fault tolerance in Open MPI using FTB has been presented [7–10] as well as demonstrated [61, 62] during several conferences and workshops. We demonstrated resilient execution of a ray-tracing application developed using POV-Ray. The application ran to completion despite a node failure by handing over the pending computational tasks to healthy processes [61]. One of the other demos 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 a MPI job sustained execution despite failures owing to the resilient runtime [62].

2.4.2 Checkpoint/Restart Library

BLCR, the Berkeley Lab Checkpoint/Restart library [119] for Linux is the one of the most prominent software available for system-level checkpointing. Several high-level libraries including MPI use it to ensure certain degree of fault tolerance in their software.

During the CIFTS project, the BLCR team integrated the BLCR with the FTB, made several improvements to BLCR to optimize checkpointing costs and successfully integrated it with a variety of Linux distributions and packages. The following provides a brief summary of the work done.

Design Optimizations and Improvements to BLCR The BLCR team has made several improvements to BLCR targeted at reducing the time and space costs normally associated with system-level checkpoints: (1) coalescing of small I/O requests into larger ones yields greater I/O efficiency; (2) in-kernel compression of checkpoint data reduces transfer times and storage requirements; (3) incremental checkpointing reduces transfer times and storage requirements by recording only the state that has changed since the previous checkpoint; (4) memory-exclusion hints enable user-space code (such as an MPI implementation) to exclude "unimportant" memory from the checkpoint (such as empty receive buffers in an MPI implementation); (5) "live-migration" moves a still-running process from one compute node to another without need for any intermediate storage; and (6) in-place rollback allows the recovery step to return an existing process to state recorded in an earlier checkpoint without the overhead of destroying the process and creating a new one. These six features will be available in a BLCR release planned for Nov. 2011, to coincide with the SC'11 conference.

BLCR-based Checkpoint/Restart Support in MPI Implementations The collaboration fostered by this project, between the BLCR development team and the development teams of the three main open-source implementations of MPI, led to several items of mutual benefit. Of particular note are (1) the work done by the MVAPICH team at Ohio State University to accelerate checkpoint I/O using buffering and (2) the *hook* interface used by Open MPI's parallel debugger support. In the first case, the work done by the MVAPICH team [70–72, 96] influenced a rewrite of the I/O code within BLCR that will appear in the next BLCR release and deliver the level of I/O performance "out of the box" that currently requires use of a buffering agent such as that implemented by the MVAPICH team. In the second case, the work done by the Open MPI team at Indiana University (IU) on integrating MPI and checkpoint/restart with a parallel debugger identified a need to allow the debugger support to interact with the checkpointing and restarting mechanism in ways not previously envisioned by the BLCR team. Together with the debugger experts, IU and LBNL designed and implemented the appropriate interfaces in BLCR to allow the required interactions [55].

FTB integration in BLCR The FTB-enabled version of BLCR has been publicly available since Jan. 2009. Support for FTB API version 0.5 was introduced in BLCR version 0.8. The current release of BLCR works with the 0.6 release of FTB, generating events for every checkpoint and restart request, providing information on the success or failure of each, and allows other components (including autonomics) to become aware of failures that may not otherwise be reported beyond a library return code. Based on feedback from the CIFTS collaborators, the next release will also include performance information in its events, allowing other components to identify bottlenecks or degradations in I/O performance. The FTB integration work in BLCR led to significant improvement in the level of useful detail provided by error and warning messages. Future work will include this information in FTB events.

Community Adoption and Ports of BLCR Cray, with support by LBNL, ported BLCR to the "CNL" kernel used on the Cray XT series and added BLCR support to their MPI runtime and job launcher. This work enables full-scale production use of BLCR on Cray XT systems. LBNL ported BLCR to the 32-bit and 64-bit PowerPC architectures running Linux, which will allow future deployment of BLCR on IBM Blue Gene systems using Linux-derivative kernels (such as ZeptoOS). To support BLCR in virtualized environments, LBNL ported BLCR to x86 and x86-64 systems running the Xen hypervisor. A contributed port to the ARM architecture enables use of BLCR in embedded and low-power environments.

The BLCR team has given several presentations [41, 43, 44, 46, 48, 97], Birds-of-a-Feather sessions [7–10], and round-table seminars [40, 42, 45, 47] on this research at various international conferences and other venues.

2.4.3 Network and Hardware Monitoring

InfiniBand has emerged as an open-standard interconnect for designing next-generation high-performance clusters. Similarly, Intelligent Platform Management Interface (IPMI) [111], which is a set of standard interfaces to monitor and manage the health of a computer system, has been gaining wide popularity. Fault data from networks and tools such as InfiniBand and IPMI is invaluable for other software running on a system that can use this data to proactively prevent catastrophic faults or make educated fault recovery decisions. The Ohio State University (OSU) team has designed and developed two independent software products to obtain this fault information from InfiniBand and IPMI and make it available to other software, through the FTB infrastructure.

FTB-IB: FTB-enabled InfiniBand Monitoring Software The FTB-InfiniBand monitoring software (FTB-IB) [66] logically consists of FTB-IB agents, which run locally on a node and monitor the InfiniBand network and publish information of faults and abnormal activities via the FTB API. The FTB-IB software is designed by using the *Asynchronous Event Handler* provided by the IB Verbs library, which is part of the Open-Fabrics Enterprise Distribution [128] software. Other system software that require fault notifications can use the FTB infrastructure to subscribe to them. The FTB-IB software publishes fault information related to InfiniBand adapter availability/unavailability, activation status of InfiniBand ports, status of InfiniBand adapter local ids and protections keys, and information on subnet manager changes.

The OSU team designed and developed the FTB-IB software and publicly released v1.0 of this software in Nov. 2008. FTB-IB version 1.0 of the software is based on the FTB API version 0.5. The FTB-IB software has been presented and demonstrated several times at international conferences and venues [68, 69].

FTB-IPMI: An FTB Interface for Intelligent Platform Management Interface The Intelligent Platform Management Interface (IPMI) [111] is a standard interface to manage a computer system independently of the operating system. The IPMI specification has been implemented by many hardware vendors. As long as the system is connected to a power source, IPMI allows out-of-band monitoring of the hardware and software status of a system and remote actions (such as system reboot)—even in cases of operating system crashes.

The FTB-IPMI is an FTB interface for the IPMI. The FTB-IPMI efficiently monitors a set of compute nodes using the IPMI interface and publishes fault events using the FTB when a problem is detected. The FTB-IPMI software relies on the GNU FreeIPMI library [110], which supports and implements IPMI for a wide range of hardware and provides a rich set of system sensor information. FTB-IPMI monitors the sensor information provided by the FreeIPMI library, analyzes this information for severity status, converts any fault information that is categorized as a “warning” or a “failure” to the FTB messages, and publishes it to the FTB. These FTB events can be caught by any other FTB-aware component to detect and/or predict failures.

The OSU team has completed the design and implementation of the FTB-IPMI software [67]. The software currently is in testing and evaluation stages at large scale. FTB-IPMI will be released in August 2011.

2.4.4 Math Libraries

Numerical libraries and dense linear algebra operations are a critical building block of many prominent science applications. Many chemistry or physics application are decomposed in a variety of problems that end up being treated by solving large dense linear systems of the form $Ax = B$. Even algorithms that are based on sparse linear algebra usually rely on the help of a dense linear algebra package to provide the computational kernels applied to the dense blocks. ScaLAPACK [124] is one of the most prominent linear algebra packages targeting the current paradigm of massive distributed-memory HPC systems. Among the kernels provided by the ScaLAPACK libraries, several key routines have been identified as needing to be refitted to become fault tolerant, because they represent a major portion of the compute time (hence are at higher risk of being impacted by failures, with the worst consequences on computation loss as a result). The LU factorization, which has good numerical stability but is costly, is often used to solve dense linear systems. The QR factorization is practical for overdetermined systems or for computing eigenvalues. The matrix-matrix multiply is also a common occurrence in production codes.

In this section, we present the design of the fault-tolerant versions of these algorithms and discuss how they benefit from their integration with the FTB layer. The FTB-enabled FT-LA software has been demonstrated at several conferences [13–16], and is a prominent example of a real use case for the FTB system.

Algorithm-Based Fault-Tolerant Routines The FT-LA software package is a dense linear algebra library that features algorithm-based fault-tolerant routines. Two releases of increasing quality have been distributed to the public on the FT-LA website [1]. With our CIFTS partners, FT-LA has been used, during the SC Bird-of-a-Feather sessions, to illustrate the benefits algorithms can harness from the coordinated recovery approach and to spark interest in the applied mathematics community. FT-LA provides checksum-based fault-tolerant algorithms, an approach that works well with standard BLAS3 kernels such as matrix-matrix multiplication or LU/QR decomposition. The original matrices are preconditioned to generate a set of larger matrices including redundancy data. During the computation, the checksum blocks are kept consistent with the data touched by the algorithm by applying the same numerical operation [117, 123]. In many cases, that means that the problem is solved with the usual algorithm, but applied on the entire matrix including the redundant checksum blocks. Should some failures occur, the FT-MPI [120] runtime rebuilds a working communicator and replaces the missing node with a new resource. The missing information can be rebuilt by applying a reduction on the data set stored on surviving processes. Since the fault-tolerant algorithm is similar to the original one, applied to a slightly larger data set, and since the recovery procedure can be expressed with a single reduction, the overall scalability is very good. We have investigated three algorithms. The GEMM kernel, which represents the matrix-matrix multiply, is a straightforward application of the core idea. Since the multiplication operation is associative and commutative, the algorithm can simply be applied to the redundant matrix. For the QR and LU factorization, the panel operation is not as simple to handle. The panel operation is not commutative (mainly because of partial pivoting), requiring a hybrid approach that relies on protecting the update portion of the matrix with ABFT checksum, and the final result (the output of the panel) with a one-time, CRC-based checkpoint. After the completion of a panel, the total checksum for the block column is computed and stored in the same memory that once was used for holding ABFT checksums, as no further updates will be made to this data. More informations on these algorithms can be found in the related publications [24, 25].

FTB Integration in FT-LA Based on the efforts in implementing the checksum-based fault-tolerant algorithms, we also focused on defining and implementing scenarios allowing for a fault-tolerant linear algebra library to benefit from cooperation with other components of the system. Technically, the integration of FTB in FT-LA is divided into two parts: the event schema and the library integration. First, we defined a generic event schema, depicting all the events generated by fault tolerant linear algebra libraries. These include events such as the ability to recover from failures, time to completion depending on the number of available spare resources should a failure occur, and notification of successful or unsuccessful recoveries. Efforts have been deployed to have the event-set exported by FT-LA to interact with the job scheduler.

We defined a set of events to monitor and react proactively to the prediction of future failures, in collaboration with autonomics components presented in previous sections. Those events have been integrated into the set of events handled by the FT-LA library. When a failure prediction is made for a particular node, the FT-LA library transfers the entire resource dataset to a replacement spare node pointed to by the migration message. This approach has several advantages compared with algorithm-agnostic proactive migration. No checkpointing is involved, and thus no performance detrimental I/O activity. The migration is built by using

MPI and therefore benefits from the maximum bandwidth. Another strong benefit is that only the threatened data are transferred, which is not the case for the classical coordinated checkpoint approach. Technically, at the end of the current computing phase, the entire dataset of the threatened node is transferred to the replacing process. When the migration is complete, the suspect node aborts, letting the replacement process step in as the new current incarnation of that particular rank. The only coordination required from the other processes is the update of their communicators. No other collective operations, computation, or data movements are involved for non-migrating processes.

Automatic Fault Tolerance in Linear Algebra Software Recently, benchmarking the performance of supercomputers has become difficult. The High Performance Linpack benchmark (HPL [118]), developed at the University of Tennessee, is the authoritative benchmark used to provide the performance data of the Top500. In recent results, HPL has seen its runtime expanding to several tens of hours (as an example, the runtime of then-#1 machine, Jaguar, was 17 hours), a fact that lies in the mathematical properties of the LU matrix factorization this benchmark features. The peak flop performance is achieved for the largest of matrix sizes and consequently the longest runtime. With HPL now being a long-running job, the probability of a failure is high. Though one can see failures disturbing the harvest of HPL results as a defect of the benchmark, another way of considering this fact is that the benchmark actually measures the usability of the platform. However, a better approach would be to have a fault-tolerant HPL benchmark to measure the effective performance achieved when factoring in the time spent dealing with the inherent unreliable nature of large systems. In the case of the HPL benchmark, transforming the algorithm in the same way that has been done in FT-LA is not an option. The HPL algorithm is the performance reference point, and changing it would be extremely detrimental to the comparative and statistical value of results over the years. Consequently, we investigated non-algorithmic ways of providing fault tolerance for the HPL benchmark. As a preliminary step, we designed an automatic fault-tolerant mechanism inside Open MPI [17]. Our approach is uncoordinated, in order to allow for nonfailed processes to continue progressing while the recovery takes place and to minimize the checkpointing noise. Inside Open MPI, we implemented an finely tuned version of the pessimistic sender-based message logging protocol [19]. Every non-deterministic event is recorded on a stable node. When a failure occurs, the recovering process follows this recorded history, which enforces the global consistency of the recovery process. The typical performance penalty of using uncoordinated middleware based recovery has been high in the past. We investigated several model and technical alterations to address those performance issues in order to reach a level of performance compatible with the task of evaluating the leadership class computing facilities [12, 18]. Two fronts have been pursued: (1) decreasing the cost of recording nondeterministic events, a goal achieved by introducing a better modeling of messages closer to the reality of zero-copy, nonblocking MPI messages and matching, and (2) by investigating different approaches to store in-transit messages, those that the restarting processes need to receive during their recovery.

2.4.5 Applications

Ultimately, the two primary "customers" of an infrastructure for fault awareness and coordinated response are the applications running on HPC systems and the people operating those systems. While approaches such as math libraries with algorithm-based fault tolerance built into them provide a largely "drop-in" approach to provide resilience to some applications, others require that more be done within the application itself in order to provide useful fault tolerance. Work on applications in CIFTS has focused primarily on exploring new FT-related capabilities that are enabled by a more detailed awareness of external fault information. In this sense, we are trying to look beyond the simple checkpoint/restart (CR) capability that is the de facto standard solution to application resilience today.

One class of application that is growing in importance scientifically is coupled multiphysics simulations, integrating components representing several types of physics into a single simulation. One such application is the Integrated Plasma Simulator (IPS), developed by the SciDAC Center for Simulation of RF Wave Interactions with Magneto-hydrodynamics (SWIM) fusion project. The ORNL CIFTS team collaborated with SWIM project researchers from ORNL and IU to demonstrate how the IPS can use external fault information to enable more intelligent responses to problems the simulation might experience and to consider the characteristics of the application that enable this approach. This work was presented at the 2011 International Conference on Computational Science [11, 98].

The IPS framework provides a whole-application checkpoint/restart capability, through which all components in the simulation are periodically checkpointed. If just one task fails, all components must be rolled back to the last checkpoint in order to recover and continue the simulation. A lighter-weight alternative is "task re-execution" (TR), in which only the failed task is rerun. We believe this kind of more localized fault recovery will be increasingly desirable as computers and applications scale out and therefore experience faults more frequently. New faults *during* fault recover operations are particularly challenging to handle correctly and robustly, and localizing the recover operations is a straightforward way of reducing those issues. Task re-execution itself is relatively straightforward but does have a disadvantage: "blind" re-execution may waste computer resources depending on the nature of the fault. Through the CIFTS FTB, the application can be connected to the computer system's monitoring subsystem to gain better awareness of what is actually going on in the system and consequently make better decisions about task re-execution. Several simple cases illustrate the approach:

- Many faults take a node out of service. In this case, task re-execution is likely to succeed *if and only if* sufficient resources are still available to launch the task.
- Out-of-memory errors or segmentation faults are examples of faults that can cause tasks to fail without taking a node out of service. While the resources remain available, the nature of these errors is such that TR is unlikely to succeed. In the out-of-memory case, it may be possible to *adapt* the application to work around the fault by relaunching the task with fewer MPI processes per node in order to allow more memory per process. Segmentation faults, on the other hand, are most commonly errors in the application software that need to be addressed by a developer. In this case, promptly aborting the simulation may be the most efficacious choice.
- Failures or severe performance degradation in I/O subsystems (due to a RAID rebuild, for example) are examples of faults that might not immediately cause a running task to fail (until it needs to do I/O on the failed file system). In this case, it may be desirable to preemptively kill affected tasks and (if possible) restart them on another file system.

Besides demonstrating the concept of "intelligent" task re-execution, another important aspect of this work is an analysis of the capabilities and (application) characteristics required to enable this kind of resilience:

- A hardware monitoring or "reliability, availability, and serviceability" (RAS) subsystem on the execution platform, which provides definitive information about faults occurring in the system
- The FTB, which provides a general connection between the hardware monitor (or other services) and the application
- Services within the Integrated Plasma Simulator application framework, including
  - An event service, which distributed fault events from the FTB to relevant services within the IPS
  - A resource manager, which uses fault information to track the availability of computer nodes within its allocation

  - A task manager, which implements policies for handling faults (e.g., re-executing failed tasks) based on configuration parameters and live fault information
  - A data manager, which supports task re-execution (e.g., resetting the contents of the component's working directory).

- The loosely coupled nature of IPS applications greatly simplifies the implementation of TR capabilities. (Each IPS task is a separate `mpiexec`, and data is exchanged via files.) Applications that use MPI between tasks would require a degree of fault tolerance within the MPI implementation. Likewise, more tightly coupled simulations would require careful consideration to ensure that task-level re-execution is feasible.

Relatively few current applications are built in frameworks like IPS that make it relatively easy to provide key services to facilitate application resilience. An alternative approach is to develop a simple environment that can wrap around an application, with little or no modification to the application itself. This approach has recently been explored in another CIFTS activity at ORNL, which has resulted in a presentation and publication already [90, 92], with a second paper in preparation. The focus of this work has been on providing mechanisms which are easily accessible to application users to guide the desired responses to fault situations ("policy management") and an examination of application "health" indicators that can easily be extracted with little or no modification to the application and used to monitor application progress.

In this work, we have developed a *Fault Coordination Framework (FCF)*, which integrates a policy manager, system monitor, application monitor, and task launcher using the CIFTS FTB as the communication mechanism. We use SEC (Simple Event Correlator) for the policy manager and Ganglia for the system monitor; the application monitor and task launcher are small, purpose-built codes. We used a molecular dynamics (MD) application to demonstrate the operation of the framework, although the FCF is application-independent apart from the application monitor.

The application is monitored by extracting diagnostic information from its output. For the MD application, we extract the (wall-clock) time per simulation step, and the computed temperature and energy of the molecular system. The user can supply policies that succinctly express conditions under which these diagnostic values are considered anomalous (e.g., a step that takes significantly longer than the average step, or values or changes in the temperature or energy that exceed reasonable amounts). Such conditions can arise, for example, because of data corruption or slowing from failing system components. The application monitor can also detect application failures for other reasons (e.g., segmentation fault or floating-point exception), and the system monitor can likewise detect hardware faults. The user may supply policies that trigger on any of these conditions, and dictate the desired response. Typical responses might be as simple as restarting from the most recent healthy checkpoint, if sufficient resources are available to the job, or aborting the simulation if not; through SEC, arbitrarily complex actions are possible by calling out to user-supplied scripts.

Our studies illustrate the benefits of systemwide fault awareness to applications in terms of the ability to differentiate among the many different kinds of faults that an application can experience, thereby permitting a more targeted and therefore more effective response. Our work with the Integrated Plasma Simulator suggests a set of services that would be necessary for application frameworks to consider providing in order to facilitate the implementation of resilience strategies beyond checkpoint/restart. The development of the Fault Coordination Framework suggests an alternative path, in which a similar set of services can be provided *external* to the application, while still allowing straightforward user control of the identification of and response to faults.

2.4.6 Resource Managers and Job Schedulers

Resource managers and job schedulers (collectively referred to as RM/JS, henceforth) , play an important role in systemwide coordinated fault tolerance. This role is based on the fact that RM/JS software typically has centralized information pertaining to all jobs and all users using the system, as well as all resources in the system. Thus, resource managers and job schedulers typically are viable candidates to receive and react to fault information. Live fault information can prove to be of immense use to RM/JS software, which can make on-the-fly decisions about scheduling and migrating jobs based on the current resource and system state. Live recovery information about formerly faulty but now recovered resources can allow job schedulers and resource managers to manipulate resource pools and change job allocation characteristics to improve the overall usability and efficiency of the system. Published information about potential migration, changed resource allocations, or changed job allocation times can, in turn, help other software such as applications and MPI make educated and optimal decisions at their end.

During the course of this project, we investigated two prominent resource managers and job schedulers: SLURM and COBALT. The sections below describe our experiences and results with this software.

Experiences with SLURM SLURM [125] is a popular, open-source resource manager and job scheduler developed by Lawrence Livermore National Laboratory. SLURM is actively managing resources on many 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 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 thus can track resource changes, job status, and SLURM failures. SLURM can then be controlled externally through its command-line tools, a library interface, or an external scripting language. FTB integration of SLURM provided us an insight 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 making 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.

Experiences with COBALT The COBALT [109] resource manager is an open-source resource manager that is used on a large number of Blue Gene/L and Blue Gene/P systems, including the "Intrepid' system at Argonne. COBALT supports several pieces of functionality in management systems, such as scheduling, queue management, hardware resource management, and process management [132].

FTB integration with COBALT was first demonstrated at Supercomputing 2007 [23]. This FTB-enabled version of COBALT subscribed to MPICH2 faults events and recovery events from diagnostic engines, analyzed those faults events, and was capable of taking reactive actions and publishing FTB events indicating the action taken. In particular, we have demonstrated [23] how, on receiving MPICH2 communication failure and node failure faults, COBALT could remove the faulty node from the hardware resource management pool, thus preventing the use of the faulty nodes for further job allocations. These FTB fault events could also be caught by diagnostic engines, which could then run diagnostic tests on the faulty node and publish a recovery message via FTB, once the recovery was completed. On receiving the recovery FTB messages, COBALT could add the node back into the resource pool. Similar functionality could be achieved when COBALT is integrated with the fault predictor and aggregator tools (described in Section 2.4.7, [104, 105])

Further planned FTB fault event integration includes adding support in COBALT to co-relate fault messages coming from different software on the same node to a single root-cause and, using that information to make a decision, adding FTB notification events indicating reservation time status, working with task managers to replace faulty allocated nodes on the fly, and notifying users and administrators (with the help of autonomic scripts) of the status (e.g., job started, job completed, allocation time exceeded, job killed) of their jobs.

2.4.7 Autonomics for Leadership-Class Machines

The ability to analyze system event data in real time and interpreting the results for easier fault analysis, diagnosis, and prediction forms the central piece of *autonomics* and is deemed indispensable as the scale and the complexity of HEC systems increase. From the inception of the project, the CIFTS team has believed that leveraging autonomic computing infrastructure would allow instantaneous dissemination of information about imminent faults, thus contributing significantly to systemwide fault awareness. Noting, however, that autonomics in leadership-class machines was still in a premature stage, the CIFTS team focused on a wide spectrum of what could constitute autonomics in such high-end systems.

Transforming RAS Event Data to Better Understand Fault Patterns To recognize impacts and implications of faults is the first step to the fault awareness. To this end, the CIFTS team has focused on understanding system failures through analysis of reliability, availability, and serviceability (RAS) event data generated by leadership computers such as the CRAY series or the IBM Blue Gene series. The volume and complexity of RAS event logs have reached the point where the manual review of this data is error-prone and inefficient. In addition, the highly irregular nature of event logs adds another dimension of ambiguity in extracting information from the data. For these reasons, CIFTS efforts have been targeted on two efforts:

- Preprocessing log entries to extract syntactic structure of the data for fault analysis
- Sifting correlations or relationships between different RAS event types to aid fault analysis

Although RAS event logs provide a rich source of information about system status, they cannot be directly used for analysis because of their highly unstructured and redundant nature. Despite the crucial role of preprocessing, previously existing techniques were often ad hoc and mainly concentrated on maximizing rates of compression. Although these techniques achieved high compression rates, they removed important failure event patterns and their traces. Also, they did not consider the fact that a single failure may be reported by multiple components in the system. Processing of such events often resulted in identifying spurious patterns that often led to wrong induction.

To address the above issues, we developed a RAS event log preprocessing methodology [106] that not only cleans the data but also extracts relevant event types and their occurrences, thus assigning syntactic structures to the data for fault analysis and modeling. The technique is performed in three tightly coupled steps: (1) event categorization using regular expressions, (2) spatiotemporal event filtering that considers context of events (event start and end times, location, count, etc.), and 3) causality-driven filtering that integrates correlated events found by association rule mining. The methodology was applied to two sets of logs that are collected respectively from Cray XT4 at Oak Ridge National Laboratory [94] and Blue Gene/L at San Diego Supercomputing Center. In particular, performance of most failure prediction models [27] were found to improve by up to 174% with a compression rate of more than 90%.

In addition, our expertise built during the course of the log processing effort led us to win *"The Cray Log Analysis contest"* at WASL'08 [107].

Tools for Context-Driven Root Cause Analysis and Anomaly Detection Events described in RAS logs are irregular in their occurrences. Around the time of a failure, an avalanche of events is generated portraying different views of the failure observed from different components. Although large, the entire set of logs generated during the span of this period are mostly redundant leaving only a fraction of data relevant for analysis. However, sifting such spurious events is by no means tractable without proper aid. When multiple components are the source of a failure, the root cause is best identified by tracing a stream of highly correlated events. However, such correlations are hard to capture unless temporal intervals, when the correlations stand out, are carefully predetermined.

To address this issue, we developed a RAS event navigation tool called *RAS data Analysis through Visually Enhanced Navigation (RAVEN)* [91]. RAVEN visually overlays various types of RAS events on a physical system map. Using these, correlations between different event types, in terms of both their counts and locations, at a given time. RAVEN was initially designed to assist users with tracing event patterns through context-driven navigation of RAS logs. By displaying the number of occurrences of an event type observed during a select time interval on the system map (which is the physical layout of the system), RAVEN provides a compact and intuitive representation of event snapshots and has proved useful in understanding faulty situations and identifying root causes of some system failures. More importantly by superimposing the displacement of user applications on top of event snapshots, events or event patterns [29] specific to a certain user application can be captured.

In many cases, entries in RAS logs describe events that have pairwise relations. For example, most Lustre log messages report failed I/O attempts between object storage servers and object storage clients. Likewise, error messages reported by the Basic End to End Reliability (BEER) protocol at the Portals layer describe failed communication attempts between compute nodes. If unusually large numbers of these messages are concentrated on very few nodes, those nodes are almost certainly in an abnormal state. Lustre messages viewed closely from a pairwise context may disclose problematic object storage targets. If data is further examined with additional context, more descriptive clues can be drawn. For example, if the set of nodes addressed by BEER messages is allocated to a single application, the fault is likely caused by the application rather than other components in the system. User applications that are not properly tuned for the intended scale tend to impose unforeseeable negative impacts. These typically involve excessive communication patterns between the nodes occupied by the application or ill coordinated checkpoint attempts. Such an abnormality is difficult to detect unless log event data is analyzed by a context-driven manner. Although RAVEN can potentially help users understand problematic situations through context-aware navigation, it is impractical to browse through the whole duration of the application run. For this reason, it has been suggested that the tool would bear more practical value by being able to pinpoint some time periods when faults are most probable. To this end, we developed a framework [93] that adopts the concept of entropy to denote time periods when either the entire system or a certain application seems to be out of its normal behavior pattern. More specifically, entropies are measured by mapping each compute node where the event occurs into the application occupying the node. Roughly speaking, an entropy value denotes how evenly the occurrences of the events observed within the current time window are scattered across all user applications. For continuous monitoring [28], entropies are measured by aggregating all events that occurred within a time window (i.e., a time interval of fixed length that spans from the present time to a certain time in the past). The CIFTS team is currently working with National Center for Computational Science to deploy the framework on the Jaguar leadership computer.

IBM BG/P Fault Analysis and Prediction Engines The CIFTS framework and FTB infrastructure were also used for gathering, understanding and analyzing failure patterns based on RAS fault event and job logs and predicting future impending faults on the Argonne Intrepid system.

Most research on fault prediction is based on examining fault events during the observation window and predicting whether a fatal event. In addition, most prediction research focuses solely on RAS logs found on large systems. This research is not sufficient for petascale systems for the following reasons:

1. Large systems such as Intrepid have several thousands of components. Thus, predicting the “location” of the potential fault becomes a critical requirement. The efficiency of corrective actions such as process migration or checkpointing is heavily dependent on predicting the correct location of the fault.

2. In addition to the location, knowing the “lead time” (i.e., time interval preceding the time of failure occurrence) becomes important in order to ensure sufficient time to perform a proactive corrective action such as checkpointing

3. In addition to location and lead time, it is important to understand the relationship between future faults and their impact on actual jobs. Thus, solely analyzing RAS logs is not sufficient. A “fatal” RAS log fault might not have any significant impact on user jobs. One needs to understand the RAS logs in collaboration with other system logs such as the job scheduler logs.

In collaboration with the Illinois Institute of Technology, we developed a FTB-enabled *fault analysis and prediction engine* for practical use on a Blue Gene/P system. The engine was designed using a genetic algorithm [129] method. We analyzed the IBM Blue Gene/P Intrepid system logs, accumulated over eight months, and refined well-known prediction metrics to include location/lead time metrics.

The BG/P failure analysis and fault prediction engines consist of the following components:

- *Rule Analyzer engine*: which is currently capable of analyzing historical events to capture the cause-and-effect relationships between fatal events and their precursor warnings

- *Fault Predictor engine*: which is currently capable of actively monitoring runtime Blue Gene/P RAS events and producing a warning when a fatal event will occur; along with the location and lead-time information

- *FTB-enabled Fault Publisher engine*: which publishes fault information to the FTB system

Corrective actions could then be taken by software that is FTB-enabled and subscribed to receive this fault information. For example, the COBALT job scheduler could receive the location of the fault and exclude the node from further allocations. Our study showed that, using location information provided by our model during fault prediction, we could reduce the time wasted due to unnecessary checkpointing by 25%–50%, depending on the data size being checkpointed [39, 65, 105].

The current fault analysis and prediction engine is FTB-enabled to publish information about location and lead time. Detailed analysis of our design can be found in [105]. This work was also presented at Supercomputing’09 [103].

Information about the impact of potential future faults is currently not FTB-enabled and is the next logical step for this research. Details on job and RAS log co-relation can be found in [104].

IBM Blue Gene FTB-enabled RAS Event Monitor/Notifier Engines In addition to our research on pre-processing RAS log entries for their syntactic structure and correlations, we have developed the FTB-enabled RAS Monitor/Notifier engine [5] that will help system maintainers and administrators filter and glean certain status information (e.g., the Intrepid administrators typically monitor all RAS events with a “failure” severity in order to determine current and potential failures) from the RAS database and be notified when such information becomes available. The availability of such an FTB-enabled engine provides a consistent and standard way for all administrators to get custom RAS information without having to write individual custom scripts and tools.

The FTB-enabled RAS Monitor/Notifier engine has two significant components:

1. *The FTB-enabled RAS Event Monitor:* The FTB-enabled RAS Event Monitor continuously tracks the RAS database on the IBM service nodes. It allows the administrator to specify a custom configuration file detailing the RAS events that the administrator is interested in monitoring. The FTB event monitor converts the target RAS event into FTB-compatible event and publishes it via the FTB system.

2. *The FTB-enabled RAS Event Notifier*: The FTB Event Notifier engine subscribes to the FTB system to receive FTB events. The engine allows the administrator to specify custom actions against specific RAS events and criteria; for example, the administrator can choose to be notified via SMS for critical events and via email for not-so-critical events. Thus, different administrators can specify their individual criteria and notification mechanisms by using a single tool.

The FTB-enabled RAS Monitor and Notifier engines are generic tools that can be tailored for any leadership-class system. The engines are split into two components to allow them to be reused on other systems with logs other than the RAS logs. On Linux clusters, the FTB-enabled Event Monitor can be customized to read Linux-specific log files and convert log entries to FTB events. The FTB-enabled Event notifier will not require any modifications.

FTB-enabled Administrative Tools The FTB can be visualized as a rich repository of live fault information, at any instant. The CIFTS team has designed a suite of tools that would be of natural use to any administrator to glean information from the FTB. This suite of tools are packaged as a part of the FTB-0.6 software and can be downloaded from the CIFTS website or wiki [4, 5].

Some of the commonly used tools are described below:

- *FTB-enabled Universal Loggers:* The FTB-enabled universal loggers are stand-alone tools that subscribe to every FTB event on the FTB infrastructure. These tools are useful for the administrator who wants to track every fault event on the system. From an implementation perspective, the FTB software currently contains two flavors of the tool: (1) a universal logger using *asynchronous subscription* mechanisms of the FTB-0.6 software and (2) a universal logger using *synchronous or polling-based subscription* mechanisms of the FTB-0.6 software.

- *FTB-enabled Watchdog Tool* The FTB-enabled watchdog tool is stand-alone software that can be used to check the presence and working condition of the FTB infrastructure on a system. This tool publishes events to the FTB system and waits to receive the same events from the system. Multiple watchdogs can be used in conjunction with one another to find the overall availability of the FTB infrastructure.

- *FTB-enabled Pingpong Tool* The FTB-enabled pingpong tool allows administrators to check the availability of a communication path between FTB software (and corresponding agents) on two distinct nodes, as well as the round-trip time for FTB communication.

- *FTB-enabled All-to-All Tool* The FTB-enabled all-to-all tool allows administrators to check the availability of communication paths on all potential nodes running FTB-enabled software and measure its communication performance.

FTB-enabled Syslog Software The Syslog protocol [122] 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. To leverage the pervasive presence of Syslog, we developed a FTB-Syslog [5] 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. Active decisions made by FTB components are also logged to Syslog for provenance. To filter events of interest from the Syslog stream, we plan to investigate developing a continuous query language 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.

3 Outreach Efforts

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 [4] provides overall status and progress of the project, (2) the CIFTS wiki [5] serves as a repository for all detailed documents for all aspects of CIFTS research, including weekly/other meetings minutes, (3) the CIFTS SVN [2] serves as a repository for code development, presentations, papers, meetings, and supporting documents, and (4) the CIFTS TRAC [3] 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 [38] 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 [7–10] 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.

4 Summary and Future Extensions to CIFTS

The CIFTS team's goals have focused on providing end-to-end fault tolerance for applications using high-end computing systems. Work has encompassed two broad aspects: (1) fault tolerance techniques and enhancements to various software components including MPI, resource managers, applications, and math libraries and (2) coordination of fault information among various components. The team has identified many challenges in providing such end-to-end fault tolerance and addressed many of them as demonstrated in the large number of publications and high-quality software that has resulted from this project.

Nevertheless, the team has identified various issues and challenges that, because of time and budgetary constraints, have not yet been addressed. Some of these challenges and future extensions to CIFTS are described in this section.

- Authentication and security: A coordinated framework such as CIFTS allows any system software, including user applications and libraries, access to systemwide fault information. While users can subscribe to receive specific information about faults (e.g., fault information only from nodes where the user job is running), nothing prohibits users from receiving all the fault information emerging from various parts of the system. Our experiences with CIFTS show that while this gives users more flexibility in understanding which faults impact their job, it causes a degree of concern for administrators. Administrators of supercomputing centers and leadership-class machines have traditionally been hesitant to share systemwide fault information with users. This concern is not without reason. Extra information, especially about faults, if provided to naive users can cause unnecessary anxiety, resulting in extra support calls and trouble tickets—which in turn increase the burden on supercomputing center personnel. Dissemination of fault information to all users, especially to naive users, also leads to concerns about the potential negative public perception of a system being unstable or fault-prone—a situation every supercomputing center wants to avoid.

  To tackle these big issues, we plan to design authentication frameworks and work on solutions that can limit which users can subscribe to what kinds of events, originating from what parts of the system. For example, our authentication frameworks would allow administrators to control dissemination of RAS [121] fault information so that only a subset of fault information can be sent to other subscribers. Apart from the technical research challenges, designing such a framework would require us to work with administrators to understand their specific data-sharing concerns and to work with application users to understand the *minimum* level of fault information necessary to make educated decisions.

- Future analysis aggregators and engines: Our runs on production systems like Argonne's Intrepid Blue Gene/P system and the ORNL Jaguar CRAY system showed that an *immense amount of fault information can be produced* by FTB-enabled software on a system. We also noticed that fault information emerging from a single FTB-enabled software source typically was insufficient to identify the root cause of a problem. For example, a faulty network link problem would cause fault event to be published by various software on all nodes connected to the faulty link. Thus, fault information would be published by the MPI library (which would publish an MPI-specific "communication failure" fault), by the application (which would publish a generic communication error), by the FTB-enabled InfiniBand network monitoring library (which would publish an InfiniBand-specific error). Moreover, we noticed that faults emerge from all levels (low hardware level, sensor level, network level) of a system. Higher-level libraries and applications cannot be be expected to understand such low-level hardware and raw network-specific faults.

  This presence of varied fault information emerging from different sources requires the presence of "fault aggregators and analyzers" to be able to (1) obtain systemwide fault information and root-cause faults and (2) translate the fault information into high-level fault semantics that high-level libraries and applications can understand and take actions for. Our work with fault analyzers and predictors [104,105] is a step in this direction. However, research in a field like fault root-cause analysis is broad and often *system*-specific, *fault scenario*-specific, and *software*-specific.

We anticipate a need for detailed depth of research to build such high-level fault engines and analyzers that can present a better picture of the fault scene at hand. This research will help answer questions of whether it is even possible to build generic fault information analyzing engines and frameworks, and, if not, how custom fault-analyzing engines and frameworks can be designed to be scalable and intelligent enough to adapt to new, future fault situations.

- Scalable and advanced policy management: Ideally, the goal of CIFTS would be to ensure that for every failure situation, a set of comprehensive actions leading to recovery from the failure take place with cooperation from all software on the system. Practical implementations of this goal, however, face various challenges. Because failure scenarios and symptoms can vary and by out of order, it is difficult to devise a predictable sequence of recovery actions, rules, or policies, especially when distributed software is involved in the recovery. We have made some progress in this direction with our work [5] on designing a *generic* rule/policy-management framework that can handle rules or policies for a majority of predefined fault situations keeping into consideration all responses from interested software on a system. Considerable research remains, however, in making the policy management software scalable for large systems and designing a manageable system for adding rules and policies for new fault situations.

- Experiences with production large-scale systems: Most high-end computing software can be made robust through rigorous testing and scalability or efficiency studies. However, some fault management software belongs to a category of software that can be fortified and made robust only when based on real experiences on real production-scale systems. This situation is especially true for software such as CIFTS. We currently see applications failing and can analyze failure information from logs. But what is needed for CIFTS is a *vast amount of experience in running live applications and software on real large-scale production systems* so that we can understand the order and sequence in which the fault information is detected and received and how it flows in the environment. It is important to test and understand through real experience the dynamics of which software is present on a runtime system when faults are received and whether their recovery actions result in the overall stability of the system.

  However, getting prototypes on production systems of large scale can be a challenging task because of the cost and risk involved. We have had limited success with this goal and this will be our prime focus as a next step.

- Involvement with fault-tolerant standards in system software: Our experiences with fault tolerance challenges in the CIFTS framework have enabled us to contribute significantly to the standardization of fault tolerance in system software—in particular, to the MPI 3.0 Fault Tolerance Standardization efforts. The CIFTS team will continue working on the state of fault tolerance in important software such as the MPI programming library.

  Other than the MPI library, not many other system software have well-defined fault-tolerant standards or specifications. Fault tolerance standardization is needed in crucial system software such as job schedulers, resource managers, applications and supporting libraries such as math libraries. The standardization of fault events through the CIFTS initiative is a step in this direction. We have had good success with the FTB fault event standardization for MPI events [100], and we plan to work with the job scheduler/resource manager, applications and math libraries community to standardize the same.

- Vendor support: The CIFTS project has received strong support from industry vendors such as IBM and CRAY. The CIFTS work on Cray and IBM Blue Gene/P machines has been published, showcased, and demonstrated at several conferences and vendor events.

The FTB API version 0.5 received considerable feedback from the HPC community and vendors. The FTB API 1.0, which is under development, incorporates this feedback.

Our next goal is to get formal vendor support for CIFTS for the FTB API 1.0 specification and make it available on all production systems.

- Broadening the CIFTS coverage: The CIFTS team consists of groups working on projects belonging to the entire spectrum of high-end computing. Over the past few years, we have worked with various external collaborators and have expanded the scope of CIFTS beyond that of the immediate team. Our current efforts include the integration of the SLURM job scheduler/resource manager, the Fault awareness Enabled Computing Environment (FENCE) [99] project, and autonomic tools for leadership-class computing facilities at Argonne and ORNL.

  Our next goal is to expand our collaborations and coverage to include other job schedulers, resource managers, monitoring software, file systems, and applications.

5 CIFTS related Publications and Presentations

[1] Fault Tolerant - Linear Algebra. `http://icl.cs.utk.edu/FT-LA`.

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

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

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

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

[6] A. Vishnu, A. Mamidala, S. Narravula, and D. K. Panda. Automatic Path Migration over InfiniBand: Early Experiences. In *Proceedings of Third International Workshop on System Management Techniques, Processes, and Services, held in conjunction with IPDPS 07*, 2007.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] David E. Berholdt. Strategies for Fault Tolerance in Multicomponent Applications. Poster at the International Conference on Computational Science (ICCS'11), June 2011.

[12] George Bosilca, Aurelien Bouteiller, Thomas Herault, Pierre Lemarinier, and Jack J. Dongarra. Dodging the Cost of Unavoidable Memory Copies in Message Logging Protocols. In *EuroMPI*, pages 189–197, 2010.

[13] A. Bouteiller. Introducing the FTB-enabled Fault Tolerant Linear Algebra Library. Demonstration in the Argonne National Laboratory booth at the ACM/IEEE SC'07 Conference, November 2007.

[14] A. Bouteiller. FTB-enabled Fault Tolerant Linear Algebra Library. Demonstration in the Argonne National Laboratory booth at the ACM/IEEE SC'08 Conference, November 2008.

[15] A. Bouteiller. FTB-enabled Fault Tolerant Linear Algebra Library. Demonstration in the Argonne National Laboratory booth at the ACM/IEEE SC'09 Conference, November 2009.

[16] A. Bouteiller. FTB-enabled Fault Tolerant Linear Algebra Library - Using Coordination to Improve Fault Tolerance. Demonstration in the Argonne National Laboratory booth at the ACM/IEEE SC'10 Conference, November 2010.

[17] Aurelien Bouteiller, George Bosilca, and Jack Dongarra. Redesigning the Message Logging Model for High Performance. *Concurrency and Computation: Practice and Experience*, 22(16):2196–2211, 2011.

[18] Aurelien Bouteiller, Thomas Herault, George Bosilca, and Jack J. Dongarra. Correlated Set Coordination in Fault Tolerant Message Logging Protocols. *Lecture Notes in Computer Science, Proceedings of the 2011 Euro-Par conference*, September 2011.

[19] Aurelien Bouteiller, Thomas Ropars, George Bosilca, Christine Morin, and Jack Dongarra. Reasons to be Pessimist or Optimist for Failure Recovery in High Performance Clusters. In *Proceedings of the 2009 IEEE Cluster Conference*, Sept. 2009.

[20] Darius Buntinas. Standardized MPI FTB events in MPICH. Demonstration at the ACM/IEEE International Conference for High Performance Computing (HPC), Networking, Storage and Analysis (SC'10), November 2010.

[21] Darius Buntinas. Scalable Distributed Consensus to Support MPI Fault Tolerance. Poster presented at EuroMPI conference, September 2011.

[22] Darius Buntinas. Scalable Distributed Consensus to Support MPI Fault Tolerance. Technical Report ANL/MCS-TM-314, Argonne National Laboratory, June 2011.

[23] Darius Buntinas and Narayan Desai. FTB-enabled MPICH interaction with COBALT process manager. Demonstration at the ACM/IEEE International Conference for High Performance Computing (HPC), Networking, Storage and Analysis (SC'07), November 2007.

[24] Peng Du, Aurelien Bouteiller, George Bosilca, Thomas Herault, and Jack Dongarra. Algorithm-based fault tolerance for dense matrix factorizations. Technical Report 253, LAPACK Working Note, July 2011.

[25] Peng Du, Piotr Luszczek, Stanimire Tomov, and Jack Dongarra. Soft error resilient qr factorization for hybrid system. Technical Report 252, LAPACK Working Note, July 2011.

[26] Fault Tolerance Working Group. Run-though stabilization proposal. `svn.mpi-forum.org/trac/mpi-forum-web/wiki/ft/run_through_stabilization`.

[27] Jiexing Gu, Ziming Zheng, Zhiling Lan, John White, Eva Hocks, and Byung-Hoon Park. Dynamic Meta-Learning for Failure Prediction in Large-Scale Systems: A Case Study. In *Proceedings of the 37th International Conference on Parallel Processing*, ICPP '08, pages 157–164, Washington, DC, USA, 2008. IEEE Computer Society.

[28] Raghul Gunasekaran, Byung H. Park, David Dillow, Galen Shipman, and Al Geist. Real-Time System Log Monitoring/Analytics Framework. In *Cray Users Group Conference*, Fairbanks, Alaska, 2011.

[29] Raghul Gunasekaran, Byung H. Park, Galen Shipman, and Al Geist. Correlating Log Messages for System Diagnostics. In *Cray Users Group Conference*, Edinburgh, U.K., 2010.

[30] 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.

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

[32] 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.

[33] 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.

[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. Talk at the Argonne Leadership Computing Facility (ALCF), May 2009.

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

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

[38] 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.

[39] Rinku Gupta, Harish Naik, and Pete Beckman. Understanding Checkpointing Overheads on Massive-Scale Systems: Analysis of the IBM Blue Gene/P System. *International Journal Of High Performance Computing Applications*, 25:180–192, May 2011.

[40] P. Hargrove. Berkeley Lab's Checkpoint/Restart (BLCR). Round Table Discussion at the Lawrence Berkeley National Laboratory (LBNL) Booth in conjunction with the ACM/IEEE International Conference for High Performance Computing (HPC), Networking, Storage and Analysis (SC'07), Nov 2007.

[41] P. Hargrove. Advanced Checkpoint Fault Tolerance Solutions. Presentation at the HPC Workshop on Trends, Technologies and Collaborative Opportunities in High Performance and Grid Computing, June 2008.

[42] P. Hargrove. Berkeley Lab's Checkpoint/Restart (BLCR). Round Table Discussion at the Lawrence Berkeley National Laboratory (LBNL) Booth in conjunction with the ACM/IEEE International Conference for High Performance Computing (HPC), Networking, Storage and Analysis (SC'08), Nov 2008.

[43] P. Hargrove. System-level Checkpoint/Restart with BLCR. Presentation at the Los Alamos Computer Science Symposium (LACSS08), Oct 2008.

[44] P. Hargrove. Berkeley Lab Checkpoint/Restart (BLCR): Status and Future Plans. Dagstuhl Seminar "Fault Tolerance in High-Performance Computing and Grids", May 2009.

[45] P. Hargrove. Berkeley Lab's Checkpoint/Restart (BLCR). Round Table Discussion at the Lawrence Berkeley National Laboratory (LBNL) Booth in conjunction with the ACM/IEEE International Conference for High Performance Computing (HPC), Networking, Storage and Analysis (SC'09), Nov 2009.

[46] P. Hargrove. System-level Checkpoint/Restart with BLCR. Presentation at the TeraGrid 2009 Fault Tolerance Workshop, Mar 2009.

[47] P. Hargrove. Berkeley Lab's Checkpoint/Restart (BLCR). Round Table Discussion at the Lawrence Berkeley National Laboratory (LBNL) Booth in conjunction with the ACM/IEEE International Conference for High Performance Computing (HPC), Networking, Storage and Analysis (SC'10), Nov 2010.

[48] P. Hargrove and E. Roman. Preempting Torque Jobs with BLCR. Presentation at the TORQUE Open Source Resource Manager Road Map and three key topic workshop in conjuction with the ACM/IEEE International Conference for High Performance Computing (HPC), Networking, Storage and Analysis (SC '10), Nov 2010.

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

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

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

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

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

[54] Joshua Hursey, Richard Graham, Greg Bronevetsky, Darius Buntinas, Howard Pritchard, and David Solt. Run-Through Stabilization: An MPI Proposal for Process Fault Tolerance. Poster presented at EuroMPI conference, September 2011.

[55] Joshua Hursey, Chris January, Mark O'Connor, Paul H. Hargrove, David Lecomber, Jeffre y M. Squyres, and Andrew Lumsdaine. Checkpoint/Restart-Enabled Parallel Debugging. *Proceedings of the European MPI Users Group Conference (EuroMPI)*, September 2010.

[56] 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.

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

[58] 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.

[59] 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.

[60] M. Koop, P. Shamis, I. Rabinovitz, and D. K. Panda. Designing High-Performance and Resilient Message Passing on InfiniBand. In *Proceedings of International Workshop on Communication Architecture for Clusters (CAC 10)*, 2010.

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

[62] 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.

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

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

[65] Harish Gapanati Naik, Rinku Gupta, and Pete Beckman. Analyzing Checkpointing Trends for Applications on the IBM Blue Gene/P System. In *Proceedings of the 2009 International Conference on Parallel Processing Workshops*, ICPPW '09, pages 81–88, Washington, DC, USA, 2009. IEEE Computer Society.

[66] Network-Based Computing Laboratory. WEBPAGE for FTB-IB Project. `http://nowlab.cse.ohio-state.edu/projects/ftb-ib/#FTB-IB`.

[67] Network-Based Computing Laboratory. WEBPAGE for FTB-IPMI Project. `http://nowlab.cse.ohio-state.edu/projects/ftb-ib/#FTB-IPMI`.

[68] X. Ouyang. Coordinated Checkpoint/Restart support in MVAPICH2: A demonstration with the NASA Parallel Benchmark Suite. Demonstration in the Argonne National Laboratory booth at the ACM/IEEE SC09 Conference, November 2009.

[69] X. Ouyang. Proactive Fault-Resilience with Process Migration in MVAPICH2: A demonstration with Tachyon. Demonstration in the Argonne National Laboratory booth at the ACM/IEEE SC10 Conference, November 2010.

[70] X. Ouyang, K. Gopalakrishnan, T. Gangadharappa, and D. K. Panda. Fast Checkpointing by Write Aggregation with Dynamic Buffer and Interleaving on Multicore Architecture. In *Proceedings of International Symposium on High Performance Computing (HiPC)*, 2009.

[71] X. Ouyang, K. Gopalakrishnan, and D. K. Panda. Accelerating Checkpoint Operation by Node-Level Write Aggregation in Multicore Systems. In *Proceedings of International Conference on Parallel Processing (ICPP)*, 2009.

[72] X. Ouyang, S. Marcarelli, and D. K. Panda. Enhancing Checkpoint Performance with Staging IO and SSD. In *Proceedings of International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI)*, 2010.

[73] D. K. Panda. Handling Reliability at the MPI layer. BOF on Reliability of High-Speed Networks, in conjunction with the ACM/IEEE International Conference for High Performance Computing (HPC), Networking, Storage and Analysis (SC'07), November 2007.

[74] D. K. Panda. Fault-Tolerant/System-Management Issues in InfiniBand. Keynote Talk at the International Workshop on System Management Techniques, Processes and Services (SMPTS), April 2008.

[75] D. K. Panda. Designing Fault Resilient and Fault Tolerant Systems with InfiniBand. Presentation at the HPC Resiliency Workshop, October 2009.

[76] D. K. Panda. High Performance, Scalable and Fault-Tolerant MPI over InfiniBand: An Overview of MVAPICH/MVAPICH2 Project. Presentation at Tsukuba University, Japan, October 2009.

[77] D. K. Panda. Supporting Fault-Tolerance in Modern High-End Computing Systems with InfiniBand. Seminar on Fault-Tolerance in High Performance Computing, Dagstuhl, Germany, May 2009.

[78] D. K. Panda. Designing Fault Resilient and Fault Tolerant Systems with InfiniBand. Presentation at the SIAM Conference on Parallel Computing, February 2010.

[79] D. K. Panda. Designing High Performance, Scalable and Fault-Resilient MPI Library for Modern Clusters. Presentation at the Institue of Computing Technology (ICT), Chinese Academy of Sciences, October 2010.

[80] D. K. Panda. Designing High Performance, Scalable and Fault-Resilient MPI Library for Modern Clusters. Presentation at the Pacific Northwest National Library (PNNL), February 2010.

[81] D. K. Panda. InfiniBand Software Networking Technologies. Presentation at the Discovery 2015 Workshop, Oak Ridge National Laboratory, July 2010.

[82] D. K. Panda. Networking Technologies for Clusters: Where do We Stand and What Lies Ahead? Keynote Talk at the International Conference on Parallel and Distributed Systems (ICPADS '10), December 2010.

[83] D. K. Panda. Networking Technologies for Exascale Computing Systems: Opportunities and Challenges. Keynote Talk at the HPC China Conference, October 2010.

[84] D. K. Panda and P. Balaji. Designing High-End Computing Systems with InfiniBand and 10-Gigabit Ethernet. Presentation at the International Supercomputing Conference (ISC), May 2010.

[85] D. K. Panda, P. Balaji, and M. Koop. Designing High-End Computing Systems with InfiniBand and 10-Gigabit Ethernet. Presentation at the International Conference on the ACM/IEEE International Conference for High Performance Computing (HPC), Networking, Storage and Analysis (SC'09), November 2009.

[86] D. K. Panda, P. Balaji, and S. Sur. Designing High-End Computing Systems with InfiniBand and High-Speed Ethernet. Presentation at the International Conference on the ACM/IEEE International Conference for High Performance Computing (HPC), Networking, Storage and Analysis (SC'10), November 2010.

[87] D. K. Panda and S. Sur. Designing High Performance, Scalable and Fault-Resilient MPI Library for Modern Cluster. Presentation at the Ohio Supercomputer Center Booth, at the ACM/IEEE International Conference for High Performance Computing (HPC), Networking, Storage and Analysis (SC'10), November 2010.

[88] D. K. Panda and S. Sur. Designing High-End Computing Systems with InfiniBand and High-Speed Ethernet. Presentation at the International Supercomputing Conference (ISC '11), June 2011.

[89] D. K. Panda, A. Vishnu, and K. Gopalkrishnan. Designing Fault Resilient and Fault Tolerant Systems with InfiniBand. Invited Poster Presentation at 2009 National Workshop on Resiliency, August 2009.

[90] Byung H. Park. Realization of User-Level Fault Tolerance Policy Management through a Holistic Approach for Fault Correlation. Talk at the IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY), 2011, June 2011.

[91] Byung H. Park, Junseong Heo, Kora Guruprasad, and Al Geist. RAVEN: RAS data Analysis through Visually Enhanced Navigation. In *Cray Users Group Conference*, Edinburgh, U.K., 2010.

[92] Byung H. Park, Thomas J. Naughton, Pratul Agarwal, David Bernholdt, Al Geist, and Jennifer L. Tippens. Realization of User-Level Fault Tolerance Policy Management through a Holistic Approach for Fault Correlation. In *IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY'11)*, June 2011.

[93] Byung H. Park, Thomas J. Naughton, Al Geist, Raghul Gunasekaran, David Dillow, and Galen Shipman. User Application Monitoring through Assessment of Abnormal Behavior Recorded in RAS Logs. In *Cray Users Group Conference*, Fairbanks, Alaska, 2011.

[94] Byung H. Park, Z. Zheng, Z. Lan, and A. Geist. Analyzing Failure Events on ORNL's Cray XT4. In *Poster at the ACM/IEEE International Conference for High Performance Computing (HPC), Networking, Storage and Analysis (SC)*, 2008.

[95] Qi Gao, Weikuan Yu, Wei Huang, and D. K. Panda. Application-transparent Checkpoint/Restart for MPI Programs over Infiniband. In *Proceedings of International Conference on Parallel Processing (ICPP)*, 2007.

[96] Qi Gao, Weikuan Yu, Wei Huang, and D. K. Panda. Group-based Coordinated Checkpointing for MPI: A Case Study on InfiniBand. In *Proceedings of International Conference on Parallel Processing (ICPP)*, 2007.

[97] E. Roman. Berkeley Lab's Checkpoint/Restart (BLCR). Presentation at the Discovery 2011: HPC and Cloud Computing Workshop, Jun 2011.

[98] Aniruddha G. Shet, Wael Elwasif, Samantha S. Foley, Byung H. Park, David E. Bernholdt, and Randall Bramley. Strategies for Fault Tolerance in Multicomponent Applications. *Procedia Computer Science*, 4:2287–2296, June 2011. Proceedings of the International Conference on Computational Science, ICCS 2011.

[99] X. Sun, Z. Lan, Y. Li, H. Jin, and Z. Zheng. Towards a Fault-Aware Computing Environment. In *Proceedings of High Availability and Performance Computing Workshop*, HAPCW, 2008.

[100] 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.

[101] X. Ouyang, R. Rajachandrasekar, X. Besseron, D. K. Panda. High Performance Pipelined Process Migration with RDMA. In *Proceedings of CCGrid 2011*, 2011.

[102] X. Ouyang, S. Marcarelli, R. Rajachandrasekar and D. K. Panda. RDMA-Based Job Migration Framework for MPI over InfiniBand. In *Proceedings of Cluser 2010*, 2010.

[103] Z. Zheng, R. Gupta, Z Lan, and S. Coghlan. FTB-Enabled Failure Prediction for Blue Gene/P Systems. Poster at the ACM/IEEE International Conference for High Performance Computing (HPC), Networking, Storage and Analysis (SC'09), Nov 2009.

[104] Z. Zheng, L. Yu, W. Tang, Z. Lan, R. Gupta, N. Desai, S. Coghlan, and D. Buettner. Co-Analysis of RAS Log and Job Log on Blue Gene/P. In *Proceedings of the 2011 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IPDPS '11. IEEE Computer Society, 2011.

[105] Ziming Zheng, Zhiling Lan, Rinku Gupta, Susan Coghlan, and Peter Beckman. A Practical Failure Prediction with Location and Lead time for Blue Gene/P. In *Proceedings of the 2010 International Conference on Dependable Systems and Networks Workshops (DSN-W)*, DSNW '10, pages 15–22, Washington, DC, USA, 2010. IEEE Computer Society.

[106] Ziming Zheng, Zhiling Lan, Byung-Hoon Park, and Al Geist. System Log Pre-processing to Improve Failure Prediction. In *Dependable Systems and Networks (DSN)*, pages 572–577, 2009.

[107] Ziming Zheng, Byung H. Park, and Zhiling Lan et al. Winner of the "The Cray Log Analysis Contest". First USENIX Workshop on the Analysis of System Logs (WASL'08), held in conjuction with the 8th USENIX Symposium on OperatingSystems Design and Implementation (OSDI'08), December 2008.

6 Other Literature Cited

[108] Advanced Message Queuing Protocol. `http://www.amqp.org/`.

[109] Cobalt job management suite. `http://trac.mcs.anl.gov/projects/cobalt`.

[110] GNU FreeIPMI. `http://www.gnu.org/software/freeipmi/index.html`.

[111] Intelligent Platform Management Interface. `http://www.intel.com/design/servers/ipmi/`.

[112] Open Source AMQP Messaging. `http://qpid.apache.org/`.

[113] OpenSAF: The Open Service Availability Framework. `http://www.opensaf.org`.

[114] The Service Availability Forum (SAF)Application Interface Specification. `http://www.saforum.org`.

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

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

[117] Zizhong Chen, Graham E. Fagg, Edgar Gabriel, Julien Langou, Thara Angskun, George Bosilca, and Jack Dongarra. Fault Tolerant High Performance Computing by a Coding Approach. In *PPoPP '05: Proceedings of the tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 213–223, New York, NY, USA, 2005. ACM Press.

[118] J. J. Dongarra, P. Luszczek, and A. Petitet. The LINPACK Benchmark: Past, Present and Future. *Concurrency Computat.: Pract. Exper.*, 15(9):803–820, 2003. DOI: 10.1002/cpe.728.

[119] Future Technologies Group (FTG). Berkeley Lab Checkpoint/Restart (BLCR). `http://ftg.lbl.gov/checkpoint/`.

[120] G. Fagg and J. Dongarra. FT-MPI : Fault Tolerant MPI, Supporting Dynamic Applications in a Dynamic World. In *7th Euro PVM/MPI User's Group Meeting2000*, volume 1908 / 2000, Balatonfred, Hungary, september 2000. Springer-Verlag Heidelberg.

[121] Gara, A. and Blumrich, M. A. and Chen, D. and Chiu, G. L.-T. and Coteus, P. and Giampapa, M. E. and Haring, R. A. and Heidelberger, P. and Hoenicke, D. and Kopcsay, G. V. and Liebsch, T. A. and Ohmacht, M. and Steinmacher-Burow, B. D. and Takken, T. and Vranas, P. Overview of the Blue Gene/L system architecture. *IBM Journal of Research and Development*, 49(2.3):195 –212, March 2005.

[122] R. Gerhards. The Syslog Protocol. RFC 5424 (Proposed Standard), March 2009.

[123] Graham Fagg and Edgar Gabriel and Zizhong Chen and Thara Angskun and George Bosilca and Antonin Bukovsky and Jack Dongarra. Fault Tolerant Communication Library and Applications for High Performance Computing. Santa Fe, NM, 2003. Proceedings of the Los Alamos Computer Science Institute Symposium.

[124] J. Choi and Jack J. Dongarra and Susan Ostrouchov and Antoine Petitet and David W. Walker and R. Clint Whaley. The Design and Implementation of the ScaLAPACK LU, QR, and Cholesky Factorization Routines. *Scientific Programming*, 5:173–184, 1996.

[125] Morris A. Jette, Andy B. Yoo, and Mark Grondona. Slurm: Simple linux utility for resource management. In *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*, pages 44–60. Springer-Verlag, 2002.

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

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

[128] OpenFabrics Alliance (OFA). OpenFabrics Enterprise Distribution (OFED). `http://www.openfabrics.org`.

[129] M. Saggar, A.K. Agrawal, and A. Lad. Optimization of Association Rule Mining using Improved Genetic Algorithms. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 4, pages 3725–3729, Oct. 2004.

[130] John Stone. An Efficient Library for Parallel Ray Tracing and Animation. In *In Proceedings of the Intel Supercomputer Users Group*, 1995.

[131] S. Vinoski. Advanced Message Queuing Protocol. *IEEE Internet Computing*, 10(6):87 –89, 2006.

[132] W. Tang. Improving Job Scheduling on Large Scale High Performance Computing Systems. In *Final Report for Starr Research Fellowship 2009*, 2009.