Final Progress Report for Project DE-FG02-02ER25536, 2004-2005
HARNESS: Heterogeneous Adaptable Reconfigurable Networked Systems
FT-MPI: Fault Tolerant Message Passing Interface

University of Tennessee, Knoxville, TN-37996-3450

Graham E Fagg
fagg@cs.utk.edu

This report is divided into a number of subsections listed as follows:
1. Brief HARNESS Overview
2. Aims of HARNESS work at UTK
3. Completed FT-MPI system overview.
4. System software improvements
5. MPI specific performance improvements
6. FT-Applications and methods
7. Additional collaboration efforts
8. Impact, Conclusions and Future goals
9. Supported students
10. Publications
11. Appendices

1 Brief HARNESS Overview

HARNESS was proposed as a system that combined the best of emerging technologies found in current distributed computing research and commercial products into a very flexible, dynamically adaptable framework that could be used by applications to allow them to evolve and better handle their execution environment. The HARNESS system was designed using the considerable experience from previous projects such as PVM, MPI, IceT and Cumulvs. As such, the system was designed to avoid any of the common problems found with using these current systems, such as no single point of failure, ability to survive machine, node and software failures. Additional features included improved inter-component connectivity, with full support for dynamic down loading of addition components at run-time thus reducing the stress on application developers to build in all the libraries they need in advance.

2 Aims of HARNESS work at UTK

The project aims at UTK were to initially support construction of applications middleware that would allow applications to be developed and used efficiently upon the new system. Thus UTK embarked upon implementing the most commonly used programming paradigm for distributed parallel computing in the form of a MPI plug-in for the HARNESS system. In addition to supporting MPI, the UTK plug-in was designed to support fault tolerant applications via modified MPI semantics as the MPI 1.2 specification did not support any failure recovery.

2.1 FT-MPI Plug-in for HARNESS

The FT-MPI system was designed to provide an application a dynamically configurable range of methods for dealing with hardware and software failures. Justification for such a system came from previous experience of building long running scalable numeric applications on large computational resources including those of the DOE ASCI project. On such large systems, failures of either hardware or software become more frequent, and constant restarting of applications becomes impractical and highly inefficient greatly reducing total through-put of such systems. Current semantics of MPI force an application to abort prematurely on a failure of one of its components. FT-MPI was designed so that an application would continue, with a range of possible recovery actions taken.

The FT-MPI system software was designed as a collection of daemons that could be run standalone in a fashion similar to PVM and LAM/MPI, or they could be compiled with extra metadata into shared object plug-ins. These plug-ins could then be loaded in any of the project partners HARNESS Virtual machines [19].

3 Project Progress at UTK

An initial C based HARNESS core and FT-MPI implementation was completed at UTK during the first HARNESS project (DE-FG02-99ER25378). As of the beginning of year 2, UTK concentrated on completing a full release of a FT MPI 1.2 specification for the ACM SuperComputing 2003 conference, validated by the Intel and MPICH test suites. Since the release of FT-MPI 1.0.0 at SuperComputing 2003, additional work was completed on improving scalability, performance, security, MPI-2 and application level support for developing and testing new fault tolerant methods. The most critical changes were released in a patched version numbered 1.0.1.

The progress covered in this report again follows the previous changes in research direction only. Many of the latest changes have resulted in quite radical restructuring of the underlying code base with approximately 60-70% of the previous code being modified in someway.

The rest of Section 3 covers details of the currently completed FT-MPI system.
Section 4 covers improvements to the system software components (i.e. the MPI runtime system and HARNESS itself). These improvements affect the general scalability of the system as a whole as well its use of OS level resources. Section 5 covers improvements to MPI specific operations (such as point to point, collective and buffer management). These improvements affect the performance of application utilizing the MPI library such as scientific applications (NWChem) built on top of numeric libraries (such SCALAPACK/Petsc). Section 6 details the design and performance of fault tolerant applications and methods. These are new applications that are designed to take specific advantage of features within FT-MPI that are not available within any other MPI implementation.

Section 7 details how the HARNESS/FT-MPI system has both been utilized by other projects, as well as any technology transfer to those other projects. Section 8 covers project Impact, Conclusions and Future work.

3.1 FT-MPI system overview

FT-MPI was formally announced and released together with a complete HARNESS runtime system at the November ACM SuperComputing 2003 conference in Phoenix, Arizona. The software distribution consisted of a complete MPI 1.2 implementation supporting Solaris, Linux, IRIX, AIX, Tru64, LinuxAlpha and MS Windows. Prior to the initial release some MPI 2 features were also added such as the C++ interface (chapter 10 of the MPI-2 specification). The software was distributed with a number of demonstration applications that covered all the possible combinations of fault handling semantics unique to FT-MPI. Prior to release the software was validated using the Intel and MPICH test suites for MPI correctness, and the SCALAPACK test suite for library compatibility. The system was released with a complete HARNESS runtime, which also included example plug-ins other than FT-MPI and comprehensive documentation. A final addition to FT-MPI prior to release was security based on SSL keys and SSL sockets so that application could mutually authenticate to resources and also authenticate commands issued externally to the runtime system based on keys issues to users.

The overall architecture of the system is shown below in figure 1. Shaded subsections belong to the HARNESS runtime system, white subsections to the MPI library (libftmpi) and SNIPE2/V is semi-hatched as it is shared between both the HARNESS and the MPI runtimes.

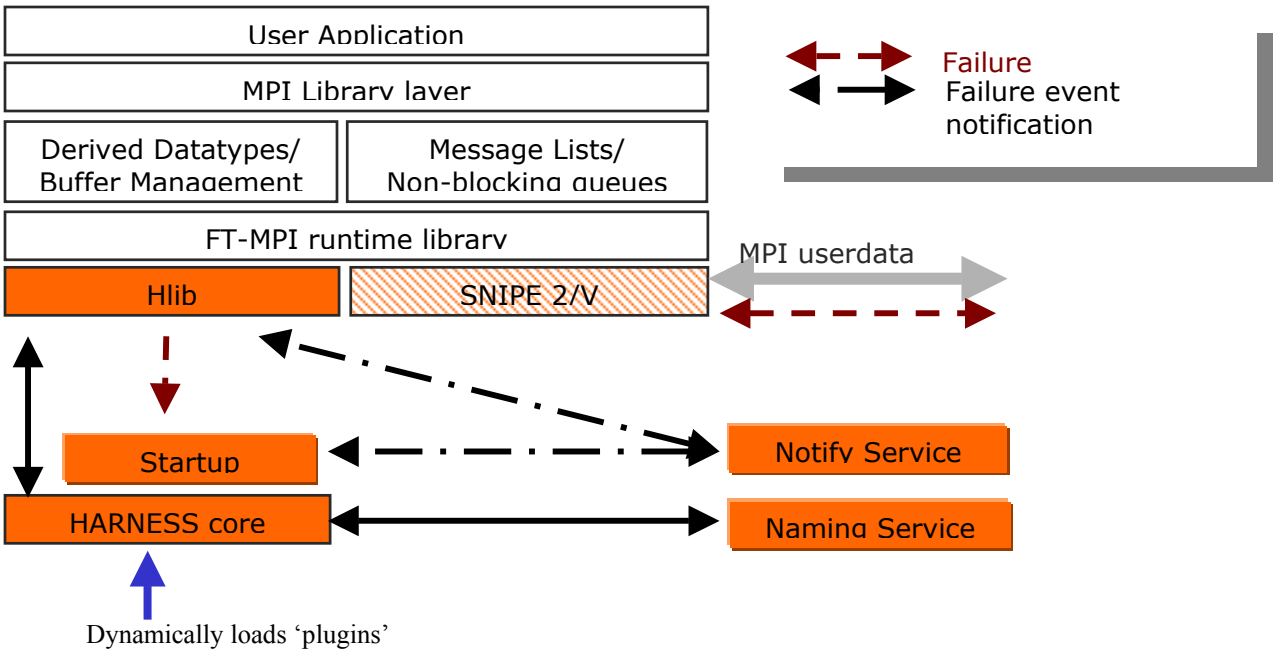


Figure 1. HARNESS/FT-MPI architecture.

4. System software improvements and additional functionality

System software is defined as any of the software glue that is not MPI specific but is required to allow an MPI based application to execute in parallel on top of some computing systems native Operating Environment. In terms of FT-MPI this includes both the FT-MPI runtime library (and plug-in) as well as all the HARNESS runtime components. User fed back as well as extensive local application testing directly prompted most of the work in this section. Improvements included a wide spectrum of potential areas such as raw performance as seen by users; primarily startup times, overall responsiveness, reduction of runtime memory footprint, scalability and robustness (reliability). Additionally the system software has been continuously updated to support new architectures (i.e. 64 bit), native runtime systems and changes in commonly used compilers, libraries, kernels etc. Also included in this section is the fault recovery mechanism within the FT-MPI runtime that allows FT-MPI to provide its FT capabilities. Many of the changes made during the last year have allowed more internal sharing of code and thus an overall reduction in both code (to maintain and verify) as well as the final build and memory costs. One example is SSL support initially developed in 2003-2004. The SSL support resulted in two separate version of the SNIPE_LITE communications library. These have since been combined, reducing both code duplication and possible errors. The final runtime system in HARNESS/FT-MPI is now quite compact at around 35 thousand lines of code, down from 70 thousand previously.

4.1 System software scalability issues

Scalability issues within the runtime system mainly focus around three areas:

(a) The use of linear algorithms to complete operations.
(b) Use of synchronous verses asynchronous communication primitives.
(c) The contention for resources at single points.

Factors (a) and (c) are usually related and their effects are usually exacerbated by the use of synchronous communications primitives, although within this project these may have been treated separately due to design and architecture considerations.

The most common issue effecting users is either the startup time for a normal MPI application, or the recovery time after the runtime system has detected an application process/task/node failure, before control is returned back to the users MPI application after the HARNESS/FT-MPI system has rebuilt a new coherent system state.

4.1.1 Startup and general responsiveness

Previously in years 2003-2004 we added an asynchronous block based method to distribute command operations between the HARNESS startup daemons responsible for process creation. During 2004-2005 we continued to refine this system as well as again enable tree based startups. To further improve startup times we also modified the way system services such as daemons like the name service interact via finite resources such as socket descriptors. Instrumentation allowed us to tune the use of these resources, and now together with a new on the wire protocol, subsystems can dynamically request disconnection transparently when they become oversubscribed. The runtime library maintains a comprehensive list of connection states and when a new communication is initiated the system transparently (& automatically) reconnects. This design had two benefits, one it did not require any code changes to the system services and two, it reduced the length of descriptor lists that had to be polled by the operating system. The polling issue was paramount as now the SNIPE library used to make connections is no longer used to transfer data, but rather the SNIPE2V library used by the MPI level messaging is now used instead, allowing the MPI runtime to poll for both user level data and system messages (failures for example) within the same operation rather than in two separate system calls. By reducing the length of descriptor lists overall polling and message latency times have been reduced. We also experimented with the new system level primitive *epoll*, but this is currently not supported widely. The effects of all of these improvements are illustrated below in figure 2, which shows the startup time for the runtime system (service daemons) itself.

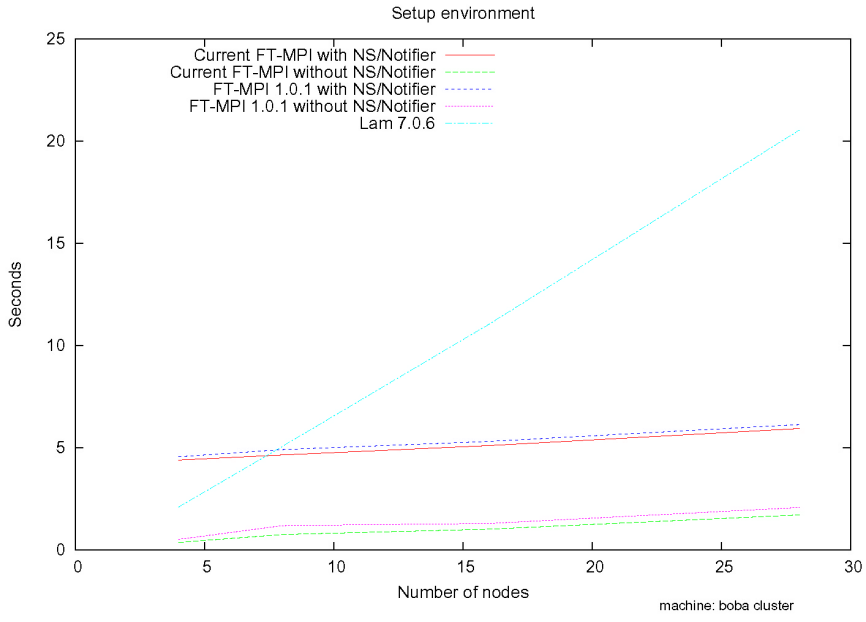


Figure 2. Startup time for the runtime system, comparing FT-MPI 1.0.1 with the final FT-MPI system and LAM 7.0.6.

The difference between 'with NS/Notifier' and 'without NS/Notifier' is that the HARNESS systems Naming and Event system daemons can be started or executed independently. The performance of the initial startup of the system using a block based asynchronous method is scalable on small clusters up to 512 nodes at which point we switch to a tree based startup mechanism. Note, this is not the time for starting an MPI job but rather the HARNESS system itself. An MPI job startup is sub one second for 128 processes.

4.1.2 System Recovery time

System recovery time is the time it takes from detecting a failure until control is returned back to a users MPI application. Control is returned when FT-MPI has built a new globally coherent MPI_COMM_WORLD, which may involve the creation of one or more application processes as well as the filtering of all MPI messages.

The FT-MPI 1.0.1 algorithm consists of a number of steps:

(a) Detecting failure and passing this to the Notifier service.
(b) The Notifier passing this event on to all MPI processes plus any other interested parties
(c) All MPI processes completing all currently started messages (where the header and/or user data has already been transmitted on the network) followed by a handshaking message flush protocol
(d) An MPI process being elected as a leader to coordinate the recovery via an atomic election IF needed
(e) The leader surveying the current system state and then building a new coherent state.
(f) The leader distributing the new state via a multiphase commit protocol
(g) MPI messages filtered depending on the new state
(h) MPI_COMM_WORLD rebuilt and network communications re-enabled
(i) Control passed back to the users application or a user provided error handler

The algorithm is recursive so that it can handle failures occurring during the recovery. For example if the leader dies, then control jumps back to the leader election and so on.

A number of the steps are either linear, use synchronous communication or are latency intolerant. In an attempt to both reduce recovery time and make it scalable across either large or multiple clusters [Grids], we have both devised a new recovery algorithm as well as changed the communication primitives in use [14]. The work involves creating multiple sub-leaders who each manage separate zones and the use of tree topologies to distribute and collect event and state changes. Figure 3 shows the effects of using various combinations of improved communication primitives. As can be seen the use of binomial tree based broadcast and gather operations during recovery improves LAN based recovery times by 50% for even small systems. It should be noted that the recovery algorithm has limits imposed on its scalability due to its use of a multiphase commit

algorithm, which requires multiple global communications even when using a multi-zone algorithm (where the zone sub-leaders still need to fully exchange and agree on a global state).

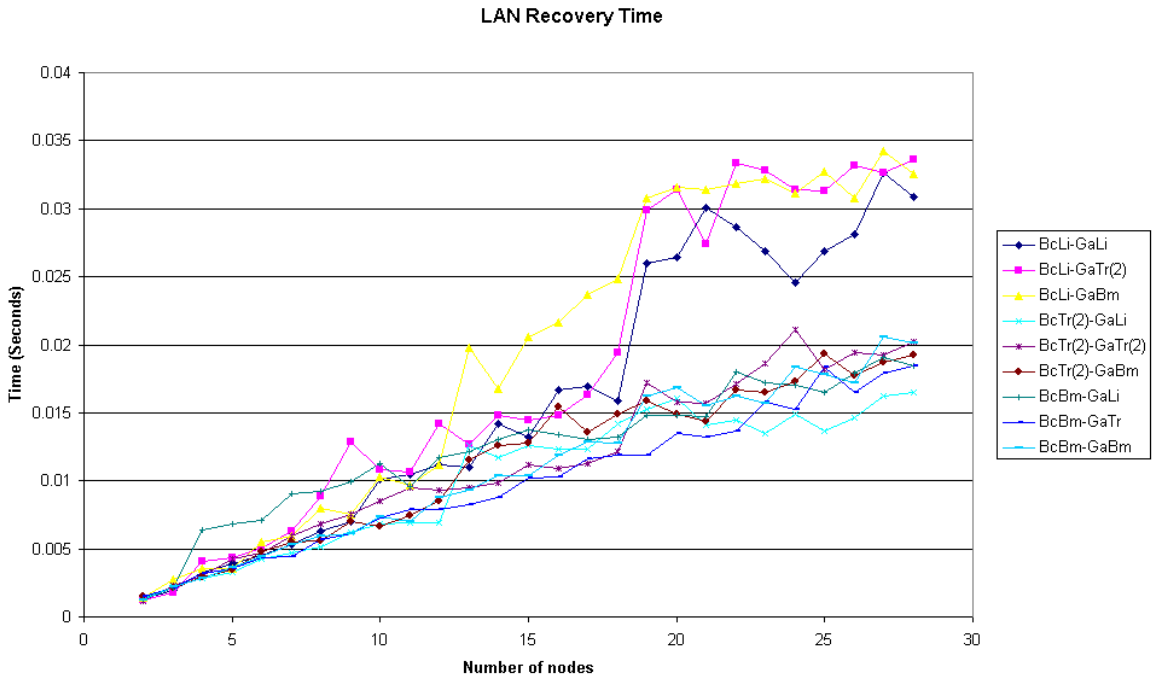


Figure 3. FT-MPI recovery times using various improved collective communication patterns. Bc = Broadcast, Ga = Gather, Li = Linear, Tr(2) = Binary tree, BM = Binomial Tree.

Figure 4 below shows how the improved communications when combined with a multi leader (or zone) approach allows for a highly scalable and latency tolerant recovery algorithm. The experiment involved the execution of a single FT-MPI application across two clusters separated by the Atlantic. One cluster was based in Tennessee and the other was located at the University of Strasbourg in France. As can be seen the multi-zone method reduces the wide area traffic and thus overall recovery times become tolerable for even distributed Grid based applications using FT-MPI.

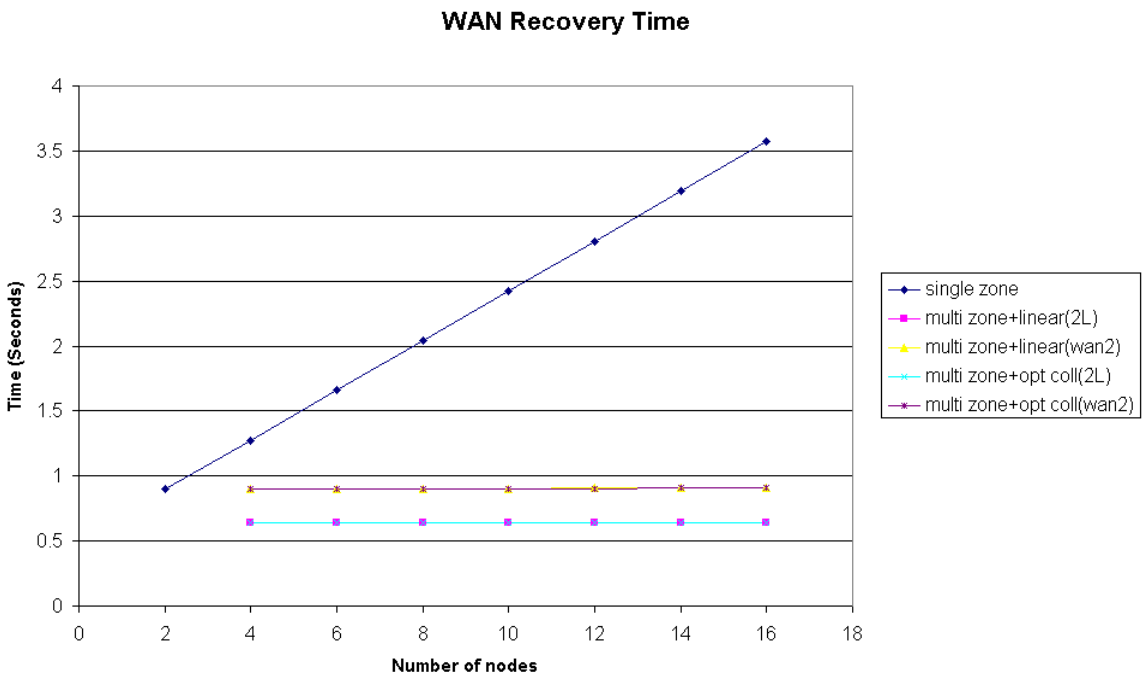


Figure 4. Multi-zone recovery for a wide area distributed [Grid] FT-MPI application executing jointly across two clusters, one located in France and the other in the US.

4.2 System software usability system

System software usability involves presenting a system that is both efficient and helpful. This includes various [user controllable] levels of verbosity and helpful (meaningful) error messages. In addition to the more common ftmpirun utility, UTK's HARNESS system provides a console similar to that found in the PVM system. The HARNESS console is more flexible and configurable, and is also designed to be driven from scripts. The console features meaningful error messages, hints, shell like history lists and line completion as well as user definable variable, argument and environment handling. Previously in 2003-2004 we started to add support for heterogeneous usage is the translation of signals from POSIX definitions to the correct local values on each target system. (I.e. a kill –X will be translated at each target correctly no matter where the user was when they issued the request). Also previously added was automatic dynamic spawning of debuggers on failures and signals if the user requested. This allowed the user to selectively debug only the processes that failed, rather than starting every process in a potentially large application *manually* in a *separate* debugger at MPI initialization. The work has been continued in 2004-2005 by both extending the options available (such as now being able to dump various core files) and also adding support for extended features of the Marmot verification tool. The console itself has also been augmented with ANSI Color support and National Language Support (NLS). Additionally the console now supports script like line history and editing on all HARNESS supported platforms.

4.3 System software build and architecture support

As new architectures come online or commonly used compilers change their features and command line arguments, so the HARNESS system also gets updated. In 2003-2004 we added our initial support for AMD64, we have recently also added support for Solaris 10 and the DARWIN systems. To further support new systems the build system has been changed so that many of the library header files are now generated automatically per target system. This was initiated by the Solaris system for which int*1 did not exist, forcing a rethinking of the complete Fortran wrappers design. Additionally the new build system also supports user overrides for *all* internally used constants.

System level support for parallel runtimes has also been expanded during 2004-2005. HARNESS process creation (used by ftmpirun) now additionally supports: PBS, Sun Grid Engine (SGE), and IBM Load-leveler.

For non-batch systems the startup system can now be instructed to use load average histories (via performance monitoring hooks within the startup daemons) to dictate the process placement in addition to the default round robin scheduling. Metrics include load average, a synthetic benchmark, memory resources and number and type of processors available.

5. MPI specific performance improvements

This section details changes to the MPI runtime library (libftmpi) aimed at improving the performance of the MPI library from the perspective of the MPI application. As in previous years the main areas of improvements were again; collective communications for both very large data and vector type operations, point to point communication operations and any operations involving complex non contiguous derived data types (DDTs). As all these areas are continuously being improved, the scope for improvements became more iterative/incremental in nature and the level of success has varied greatly.

5.1 Collective communications library

Collective communications are heavily relied upon in many applications and numeric libraries. Developers of such libraries expect that all the collective operations are tuned and that it is always faster to use a built in MPI collective function rather than implement one using explicit MPI point to point calls. During development of the FT-MPI implementation particular attention was paid to implementing collective communications as a special class of communications rather than something built on optimized point-to-point (p2p) communication. This has led to the development of a self tuning collective communications library that can provide optimal performance on any given MPP/cluster system via micro benchmarking a number of possible implementations and then selecting the fastest. The tuned collectives library within FT-MPI has also extended into a separate collective communication package (OCC) that can be used with any MPI implementation.

The main thrusts of improvements with FT-MPI collective communications during 2004-2005 has been in:

(a) Addition of new algorithms, topologies, scheduling and segmentation
(b) Improvements to the benchmarking system to reduce total test/tune time
(c) New mathematical models used to verify performance prediction models
(d) Verification of MPI collective correctness
(e) Verification of FT-MPI collective semantics during failure-recovery

5.2 New methods for collective communications

The main change in 2004-2005 for collective communication has been that all possible operations now support internal message segmentation. Message segmentation allows for algorithms to utilize maximum bisectional bandwidth by breaking larger messages into smaller sets of messages. This enables more parties to concurrently communicate. While this has increased performance for many operations the extra burden on the communications infrastructure may affect some system configurations negatively. For these operations we have also added a number of scheduled algorithms, i.e. algorithms that deliberately either limit the maximum number of communications or they force additional synchronization steps to reduce contention. The choice between which segmentation, scheduling and topology to use is still decided by a decision function during each call of an MPI collective communication operation.

Collective now supporting message segmentation include:

- MPI Broadcast: Linear (Regular and Scattered), Pipeline, Binomial, Binary (Regular and Splitted), General (n-array) Tree, Multiple Chain (single level tree combined with pipelines).
- MPI Scatter: Linear, Binomial (for small messages).
- MPI Reduce: Linear, Pipeline, Binomial, Binary, General Tree & Multiple Chain
- MPI Alltoall : Linear, Scheduled, Pairwise exchange, Ring and Bruck
- MPI Allreduce: Linear, Pipeline, Binomial and Binary

The MPI Barrier operation has also been extended to support new versions of the Bruck, and Recursive doubling algorithms. MPI Broadcast now additionally supports a new asynchronous scatter type algorithm. The algorithm detailed below in figures 5 and 6 appears to works well for some system configurations for medium sized messages over 10Mbytes yielding an almost 20% improvement over the previous optimal. Note that the MPI Broadcast decision function only selects this method when its performance is optimal. I.e. as new methods are added to the collective implementations each collective is re evaluated and its runtime decision function is updated accordingly.

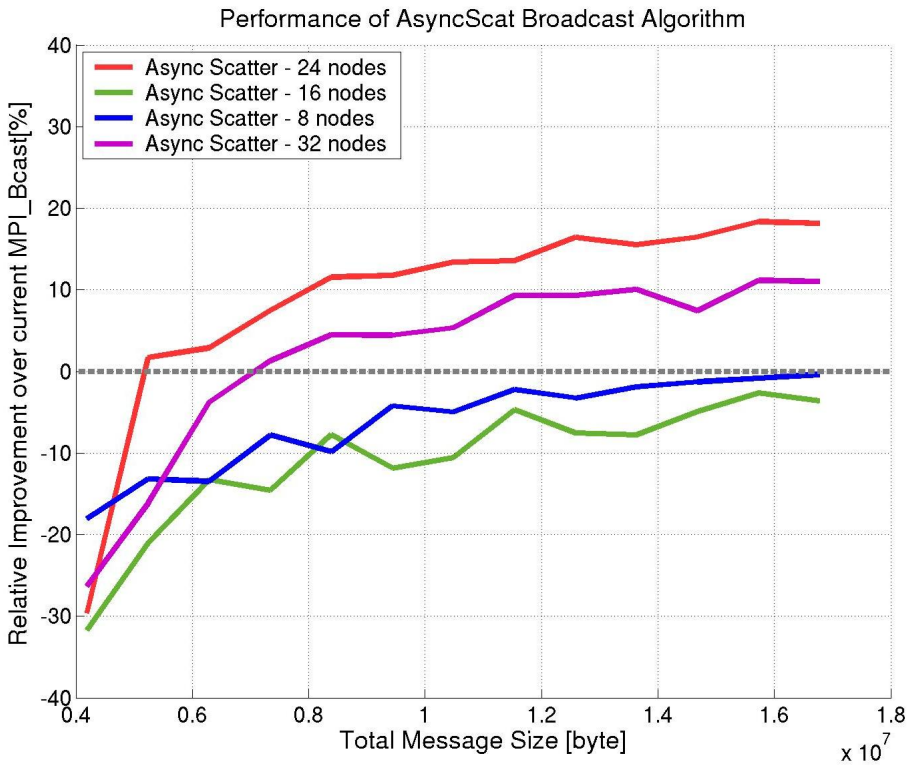


Figure 5: Performance of asynchronous scatter broadcast *relative to best of all previous* broadcast implementations.

Figure 6: Asynchronous scatter based FT-MPI Broadcast communication.

5.1.1 Collective operations during failure recovery

FT-MPI semantics requires that collective communications obey the default failure communication modes, i.e. NOP and CONTINUE. More specifically on failure the collectives should behave in a predictable fashion expected by the user even if the failure occurs while some processes are inside the collective, other have yet to join and other may have already completed the operation. To verify this we constructed a special test suite (do_coll_reset in the ftmpi/tests/general directory) that tests every MPI collective operation for each of the possible communicator modes (BLANK, SHRINK, REBUILD) both with and without MPI user supplied error handlers by tightly looping while randomly killing processes. The test suite was able to find several cases where timing issues resulted in inconsistent behavior. These required changes in both the runtime recovery runtime library as well as fixes in the FT-MPI collectives. After further testing with this suite (on the order of ten thousand cycles) we are now highly confident of the correctness of the system. The changes to the collective implementation include a switch that forces subsequent collectives to use fault tolerant linear topologies on detecting an error until a recovery phase is initiated.

5.1.2 Predicting Collective Performance

The performance of any given MPI collective operation is predicted at runtime by a decision function that uses information deduced from the MPI call (number of processes, root process, message size and layout) and the state of the system (network interconnections available) to decide which collectives algorithm, topology and segmentation size to use. The decision functions themselves are built using a test suite (OCC) that runs a number of tests exhaustively for a range of systems. These tests are quite time consuming, although it is possible for a user to do the same testing to tune their collectives. We have augmented the OCC system with a number of models of the collective algorithms within FT-MPI so that we can predict each collectives performance based on a set of parameterized constants that are much quicker to measure such as bandwidth and latency, logP and PlogP values. The different models have demonstrated varying levels in accuracy of predictions as shown below in figure 7 for the MPI Broadcast operation. The colors denote the choice of which method and parameters was chosen. As can be seen both the logP and PlogP models reasonably predict the optimal method to use for moderate to large size messages for medium to large numbers of processors. Details of the actual error compared to the exhaustively measured results are given in Appendix 11.A. Although not perfect these models provide users a good initial point from which to further tune their collectives operations on their own systems.

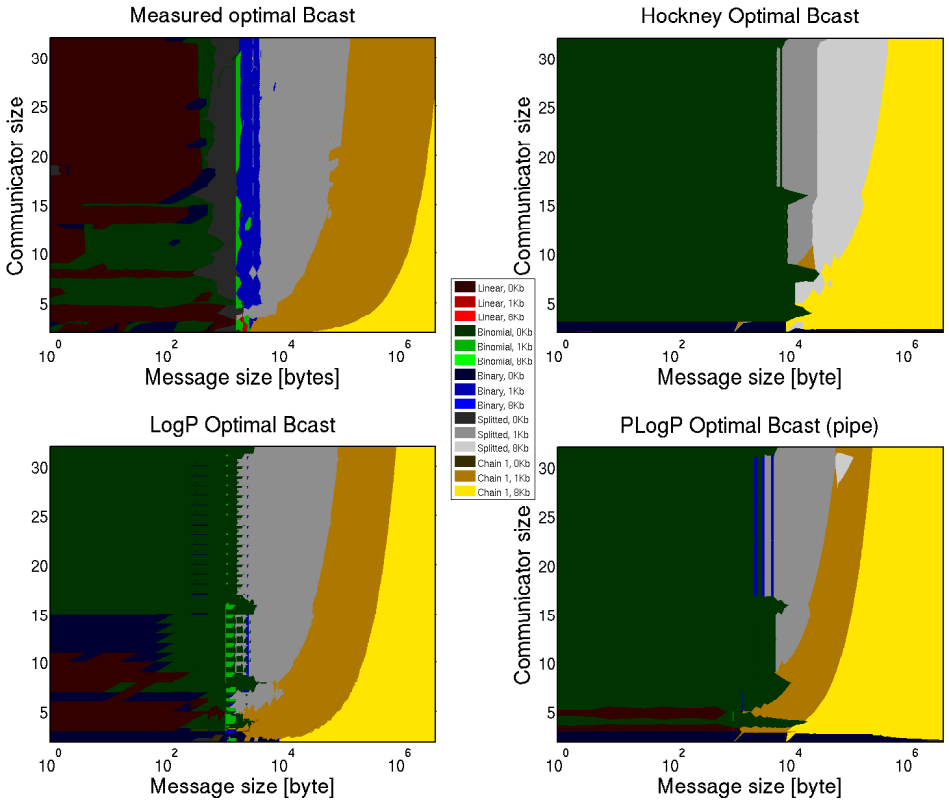


Figure 7: Fast model based prediction of optimal methods and parameters for the MPI Broadcast collective compared to the exhaustively measured optimal.

5.2 Point to point messaging performance

The initial FT-MPI release used an asynchronous message engine built on top of the SNIPE2 progress engine. This engine, although allowing good flexibility in managing multiple connections and MPI progress semantics was only able to progress a single MPI request (send and receive) operation per channel per direction at any one time. During 2003-2004 we modified this library to allow multiple MPI requests to be concatenated reducing the number of system calls resulting in a 5% improvement in latency, and better midrange (up to 128KB) message bandwidth. In 2004-2005 we continued the tuning of this engine by further integrating it with the DDT engine detailed below. The system now completely pipelines all communication operations, i.e. while sending /receiving data the system is also performing data translations, and memory operations concurrently. Which has further reduced latency by another 8%. As detailed in the runtime improvements section we have also combined the SNIPE_LITE operations with the MPI level message transfer operations of the SNIPE2V library allowing us to remove additional system calls from the critical inner communications loop while still retaining our ability to poll for external events such as process failure notifications. These improvements in small message throughput have enabled more collectives to utilize message segmentation, improving user level application performance.

5.3 Derived Data Type management

The MPI specification allows for complex messages made from recursive strongly typed data structures to be sent from one process to another. Many MPI implementations only handle contiguous blocks of data efficiently rather than these new user Derived Data Types (DDTs). We developed a library that handles these data types efficiently by using a number of advanced buffer management functions that dynamically compress data, perform scatter/gather DMA and avoids un-need memory copies. The FT-MPI 1.0.0 version handled some DDT cases very well, but other cases as shown by the HPL benchmark showed that combination of packing a completed data type into a single contiguous buffer prior to sending led to performance degradation for high block counts. In 2003-2004 we implemented a new DDT engine that allows us to mange the messaging layer directly. In effect rather than packing into a contiguous buffer we pack only as much as the point to point layer can send, i.e. dynamic flow based segmenting of conversion and transmission. In 2004-2005 this code was again rewritten to take into account both the point to point layer and the local architectures memory hierarchy, i.e. memory copy performance, cache sizes etc. This rewrite was prompted by careful profiling of FT-MPI performance for a number of applications including HPL and SCALAPACK. The performance of the final DDT engine on a deliberately non-cache friendly data type is shown in figure 8.

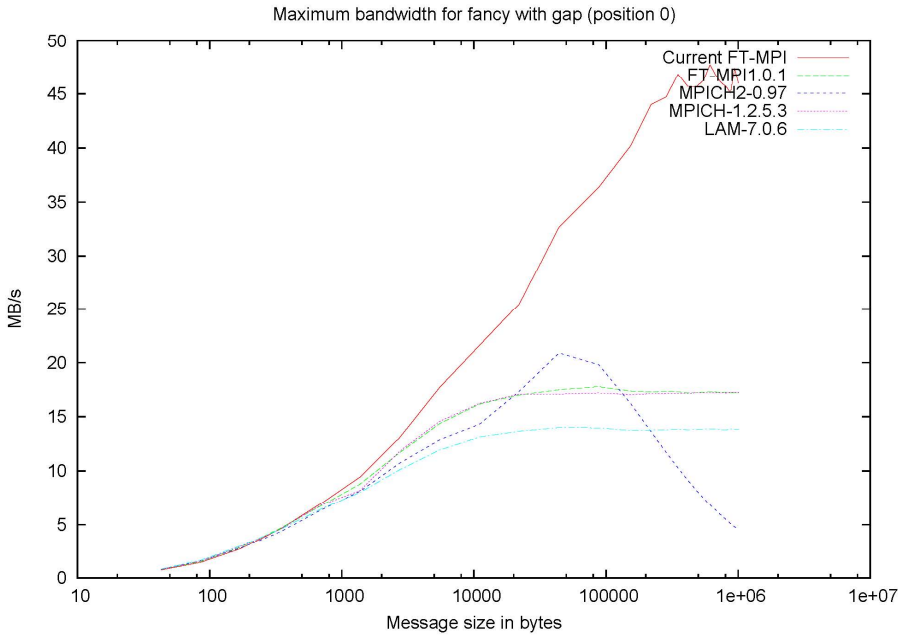


Figure 8. Performance of FT-MPI final DDT engine for a large count of small sparse user data types compared to other MPI implementations.

Figure 9 below shows the performance of FT-MPI for the trapezoid MPI indexed data structure used within HPL. Figure 10 illustrates how the combining of FT-MPI SNIPE2V

communications engine with the DDT engine improves performance for even contiguous datatypes on a newly supported Apple Darwin system.

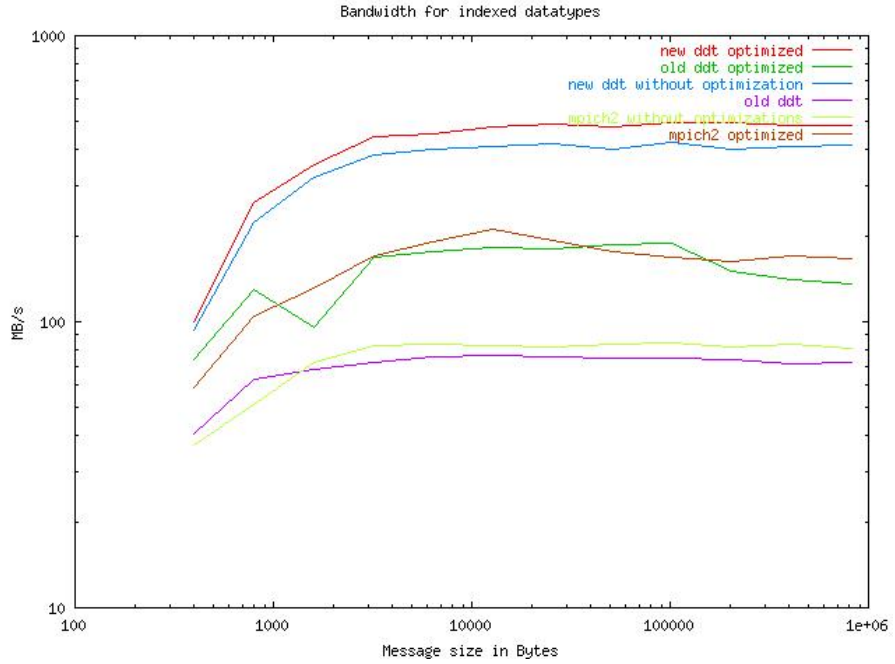


Figure 9. HPL style indexed datatype performance of various MPI implementations using different levels of pipelining operations.

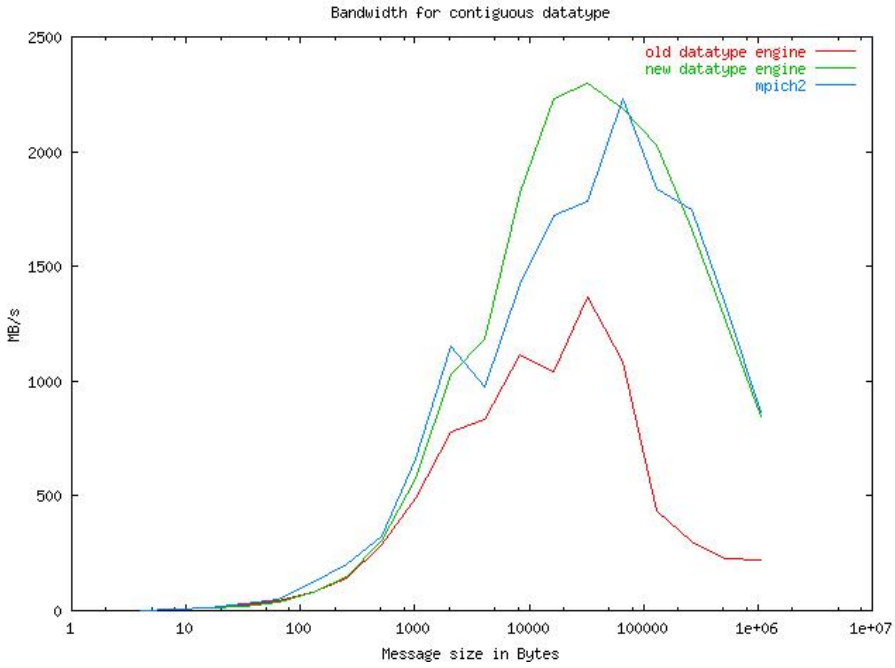


Figure 10. Performance of the new FT-MPI DDT engine for contiguous data types on a Apple Mac dual Power 5 Darwin based system compared to the release FT-MPI and current MPICH-2 implementations.

5.4 Stability and correctness

The stability and correctness of FT-MPI is constantly verified as changes are made to both the HARNESS runtime services and the FT-MPI runtime library (libftmpi). These comprise of both community accepted MPI tested suites such as the Intel, IBM and MPICH tests as well as many specific to the fault tolerant features of FT-MPI. In 2004-2005 we included a new continuously executing test that generates large volumes of point to point data in a ring topology (tests/general/ringdtc) using various complex user derived data types, while randomly terminating processes ungracefully (via SIGKILL). This suite tests the DDT engine, SNIPE2V engine, message filtering on recovery and recovery systems extensively. To date the test as executed over a one hundred thousand continuous process terminations as part of a single application run without losing messages or locking up the FT-MPI system.

6 Fault tolerant applications and methods

FT-MPI introduced new models for programming message passing systems to allow for a range of failure recovery options. During 2003-2004 these models were demonstrated for a number of numeric applications based on differing techniques for performing the actual checkpoint. These included: Diskless checkpointing by using data coding approaches, neighbor based diskless checkpointing and none checkpoint fault tolerance via modified numerical methods using data coding of replicated information in the forward path of the computation. This work has continued in 2004-2005 [6,7,8,9,10,11,12]. Additionally a CHESSI digital signal processing (SIP7) code has also been converted to use unique features of FT-MPI for enhanced Fault tolerance under the direction of the US Airforce Research Laboratory [13]. This application utilizes FT-MPIs ability to replace failed processes without completely restarting or reloaded from a checkpoint, thus maintaining time to solution guarantees.

7 Additional collaboration efforts

The FT-MPI plug-in was designed to work effectively on any of the HARNESS cores. This is achieved by a dual build system where construction of all FT-MPI plug-in services can be either built as UTK format plug-ins or as self contained daemons for use on other Cores. When the plug-ins are build as daemons, the other HARNESS cores only to have execute them, using fork-exec under ORNL core, or a ‘system.load’ under the Emory Java core. Once started the daemon based plug-ins then interoperate within themselves. At the end of 2003 during a collaboration visit by the team from Emory to UT Knoxville, the FT-MPI service and startup daemons were successfully executed under the H2O framework from Emory. Under this framework, authentication and resource management benefits are passed from H2O to the FT-MPI users environment. This work continued during 2004-2005 [19] and resulted in a full integration of the Emory University H2O system and FT-MPI. Benefits for FT-MPI included H2Os transparent support for operation across firewalls and NATs, as well as H2Os user level

file staging. During 2004-2005 Emory University in collaboration with the University of Antwerp, Belgium also worked on providing enhanced state management for FT-MPI through a replacement HARNESS Name Service system [20], this work is still ongoing but aims to further FT-MPIs Grid capabilities.

Open MPI is an open source community MPI implementation led by a core group of MPI implementers including Los Alamos National Laboratory (LA-MPI), University of Tennessee Knoxville (FT-MPI) and Indiana University (LAM/MPI). The FT-MPI team has used much of its experience in adding value to the Open MPI project, this has included direct technology transfer from the HARNESS/FT-MPI project to Open MPI. Examples include; the Open MPI runtime system (Open RTE) which was modeled on UTKs HARNESS implementation [21], the pipelining communications and DDT engine and some of the FT-MPI and OCC collective communications methods in the form of a Open MPI tuned collectives module. Figures 11 shows the relative performances of FT-MPI, Open MPI and MPICH-2 for an MPI Reduce operation on a single cluster system. Appendices 11.B contains additional results.

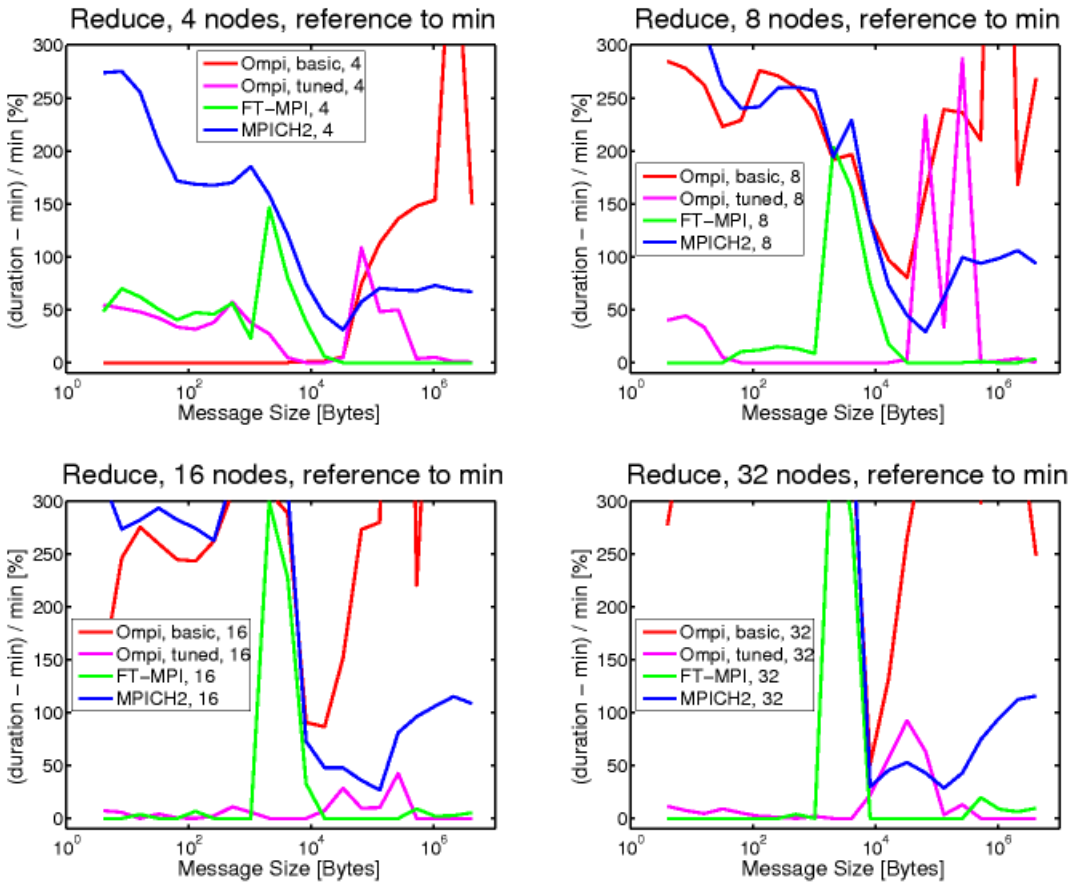


Figure 11. Performance of FT-MPI, the default Open MPI collectives, the UTK developed Open MPI 'tuned' collectives and MPICH-2 for an MPI Reduce operation. The Y axis is relative to the performance of the optimal.

8 Impact, Conclusions and Future Goals

The HARNESS/FT-MPI system has opened up new possibilities to flexibly handle failures from within high performance MPI applications. The development of the HARNESS/FT-MPI system has led to advances in system software, networking, performance modeling, and MPI implementation internals. Using the FT-MPI system has allowed researchers to develop multiple new techniques for performing, distributing and recovering application state using various checkpoint recovery schemes. In some cases these have even led to the development of new mathematical and algorithm methods.

Although the FT-MPI system is user friendly, very efficient, and faster in many areas than other contemporary open source MPI implementations it has not been widely adapted as a *vanilla* MPI system. Many users surveyed have reported that they thought it was only usable for special fault tolerant cases, even though we have published as a group numerous papers to the contrary. All is not lost as new users of the special FT capabilities continuously press us to add features. As much of the non fault tolerant performance features of the FT-MPI system has already been transferred to the Open MPI project, the lesions learnt are not lost and are instead now being presented to a wider audience. As FT-MPI is still being developed and supported users are encouraged to continue their experiments into developing new fault tolerant applications and methods.

The future aims and goals of FT-MPI are currently the same as they originally were, that of providing a stable platform to develop new methods for handling application and system failures that are expected in the futures massively parallel Peta-scale computing systems. Already work is progressing on migrating the FT-MPI process level fault tolerance into the OPEN-MPI project.

9 Students Supported

1. Thara Angskun: DVM console, system scalability, network fault tolerance
2. Jelena Pjesivac-Grbovic: Tuned collectives methods, communication modeling
3. Zizhong Chen: fault tolerant numeric methods

10 Publications

(1) Fagg, G., Dongarra, J. "Building and using a Fault Tolerant MPI implementation," International Journal of High Performance Applications and Supercomputing, 2004.

(2) Fagg, G., Gabriel, E., Bosilca, G., Angskun, T., Chen, Z., Pjesivac-Grbovic, J., London, K., Dongarra, J. "Extending the MPI Specification for Process Fault Tolerance on High Performance Computing Systems," Proceedings of ISC2004, Heidelberg, Germany, June 23, 2004.

(3) Fagg, G., Gabriel, E., Chen, Z., Angskun, T., Bosilca, G., Pjesivac-Grbovic, J., Dongarra, J. "Process Fault-Tolerance: Semantics, Design and Applications for High Performance Computing," International Journal for High Performance Applications and Supercomputing, April, 2004.

(4) Vadhiyar, S., Fagg, G., Dongarra, J. "Towards an Accurate Model for Collective Communications," International Journal of High Performance Applications, Special Issue: Automatic Performance Tuning, Volume 18, no. 1, pp. 159-167, Spring, 2004.

(5) Bosilca, G., Chen, Z., Dongarra, J., Eijkhout, V., Fagg, G., Fuentes, E., Langou, J., Luszczek, P., Pjesivac-Grbovic, J., Seymour, K., You, H., Vadhiyar, S. "Self Adapting Numerical Software SANS Effort," IBM Journal of Research and Development, June, 2005.

(6) Bosilca, G., Chen, Z., Dongarra, J., Langou, J. "Recovery Patterns for Iterative Methods in a Parallel Unstable Environment," University of Tennessee Computer Science Department Technical Report, UT-CS-04-538, 2005.

(7) Bosilca, G., Dongarra, J., Fagg, G., Langou, J. "Hash Functions for Datatype Signatures in MPI," Proceedings of 12th European Parallel Virtual Machine and Message Passing Interface Conference - Euro PVM/MPI, Di Martino, B. et al. eds. Sorrento (Naples), Italy, Springer-Verlag Berlin, LCNS 3666, pp. 76-83, September 18-21, 2005.

(8) Chen, Z., Dongarra, J. "Algorithm-Based Checkpoint-Free Fault Tolerance for Parallel Matrix Computations on Volatile Resources," University of Tennessee Computer Science Department Technical Report, UT-CS-05-561, 2005.

(9) Chen, Z., Dongarra, J. "Numerically Stable Real Number Codes Based on Random Matrices," The International Conference on Computational Science, Atlanta, GA, LNCS 3514, Springer-Verlag, May 22-25, 2005.

(10) Chen, Z., Dongarra, J. "Condition Numbers of Gaussian Random Matrices," SIAM Journal on Matrix Analysis and Applications (to appear), 2005.

(11) Chen, Z., Dongarra, J. "Condition Numbers of Gaussian Random Matrices," University of Tennessee Computer Science Department Technical Report, UT-CS-04-539, 2005.

(12) Chen, Z., Fagg, G., Gabriel, E., Langou, J., Angskun, T., Bosilca, G., Dongarra, J. "Fault Tolerant High Performance Computing by a Coding Approach," Proceedings of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Chicago, Illinois, June 15-17, 2005.

(13) Cronk, D., Fagg, G., Emeny, S., and Tucker, S. "Dynamic Process Management for Pipelined Applications," Proceedings of the HPCMP Users Group Conference 2005, Nashville, TN, IEEE, June, 2005.

(14) Fagg, G., Angskun, T., Bosilca, G., Pjesivac-Grbovic, J., Dongarra, J. "Scalable Fault Tolerant MPI: Extending the Recovery Algorithm," Proceedings of 12th European Parallel Virtual Machine and Message Passing Interface Conference - Euro PVM/MPI, Di Martino, B. et al. eds. Sorrento (Naples) , Italy, Springer-Verlag Berlin, LCNS 3666, pp. 67, September 18-21, 2005.

(15) Gabriel, E., Fagg, G., Dongarra, J. "Evaluating Dynamic Communicators and One-Sided Operations for Current MPI Libraries," International Journal of High Performance Computing Applications (to appear), 2005.

(16) Pjesivac-Grbovic, J., Angskun, Bosilca, G., Fagg, G., Gabriel, E., Dongarra, J. "Performance Analysis of MPI Collective Operations," Cluster Journal (submitted), September, 2005.

(17) Pjesivac-Grbovic, J., Angskun, T., Bosilca, G., Fagg, G., Gabriel, E., Dongarra, J. "Performance Analysis of MPI Collective Operations," 4th International Workshop on Performance Modeling, Evaluation, and Optmization of Parallel and Distributed Systems (PMEO-PDS '05), Denver, Colorado, April 4-8, 2005.

(18) Vadhiyar, S., Fagg, G., Dongarra, J. "Towards an Accurate Model for Collective Communications," ICL Technical Report, ICL-UT-05-03, 2005.

(19) Dawid Kurzyniec, Vaidy S. Sunderam, “Combining FT-MPI with H2O: Fault-Tolerant MPI Across Administrative Boundaries”. 19th International Parallel and Distributed Processing Symposium (IPDPS 2005), Denver 2005.

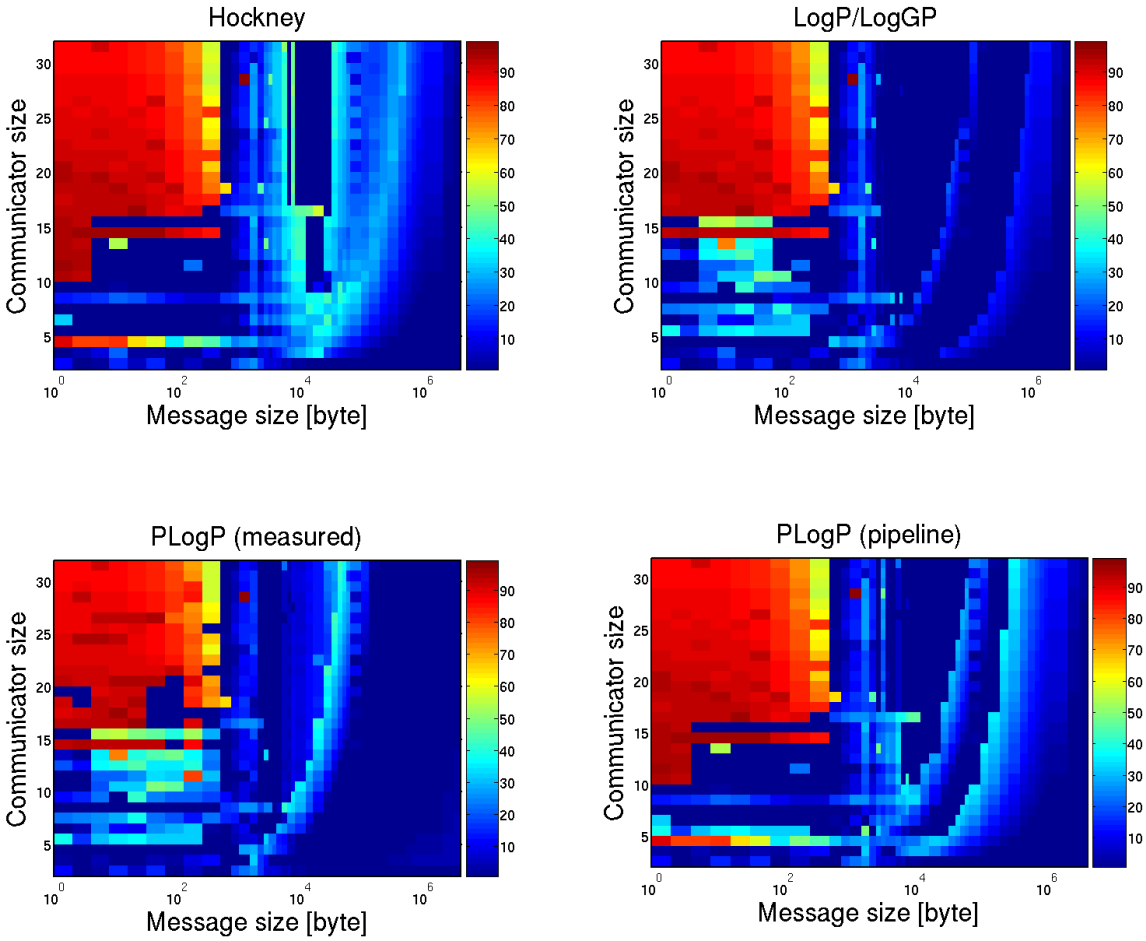
(20) David Dewolfs, Dawid Kurzyniec, Vaidy S. Sunderam, Jan Broeckhove, Tom Dhaene, Graham E. Fagg: “Applicability of Generic Naming Services and Fault-Tolerant Metacomputing with FT-MPI”, Proceedings of 12th European Parallel Virtual Machine and Message Passing Interface Conference - Euro PVM/MPI, Di Martino, B. et al. eds. Sorrento (Naples) , Italy, Springer-Verlag Berlin, LCNS 3666, pp 268-275, 2005.

(21) Ralph H. Castain, Timothy S. Woodall, David J. Daniel, Jeffrey M. Squyres, Brian Barrett, Graham E. Fagg: “The Open Run-Time Environment (OpenRTE): A Transparent Multi-cluster Environment for High-Performance Computing”, Proceedings of 12th European Parallel Virtual Machine and Message Passing Interface Conference - Euro PVM/MPI, Di Martino, B. et al. eds. Sorrento (Naples), Italy, Springer-Verlag Berlin, LCNS 3666, pp 225-232, 2005.

11 Appendices

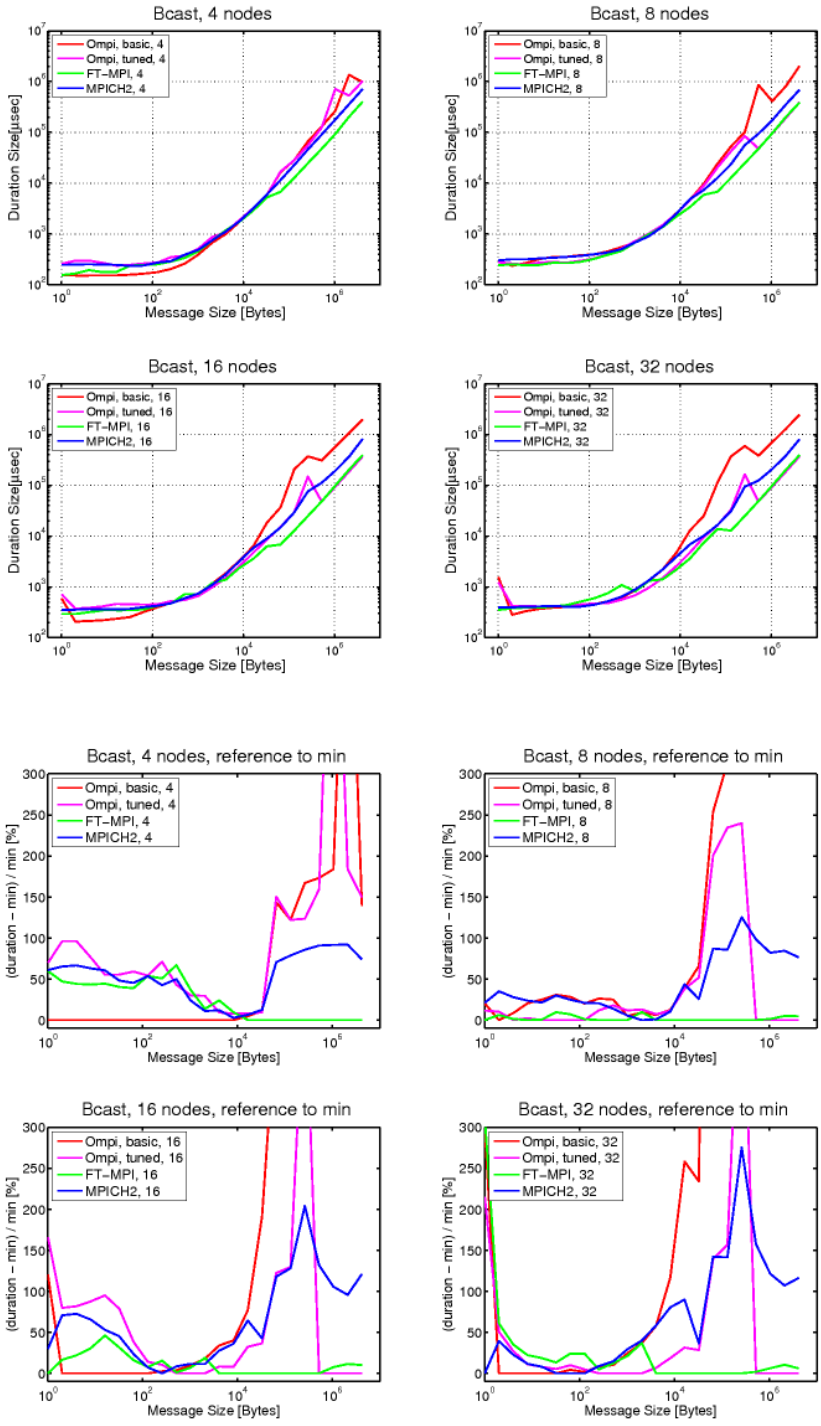
11 Appendix A

The below figure shows the error between the empirically measure results of a collective broadcast verses those predicted by a number of commonly accepted point to point communication models.



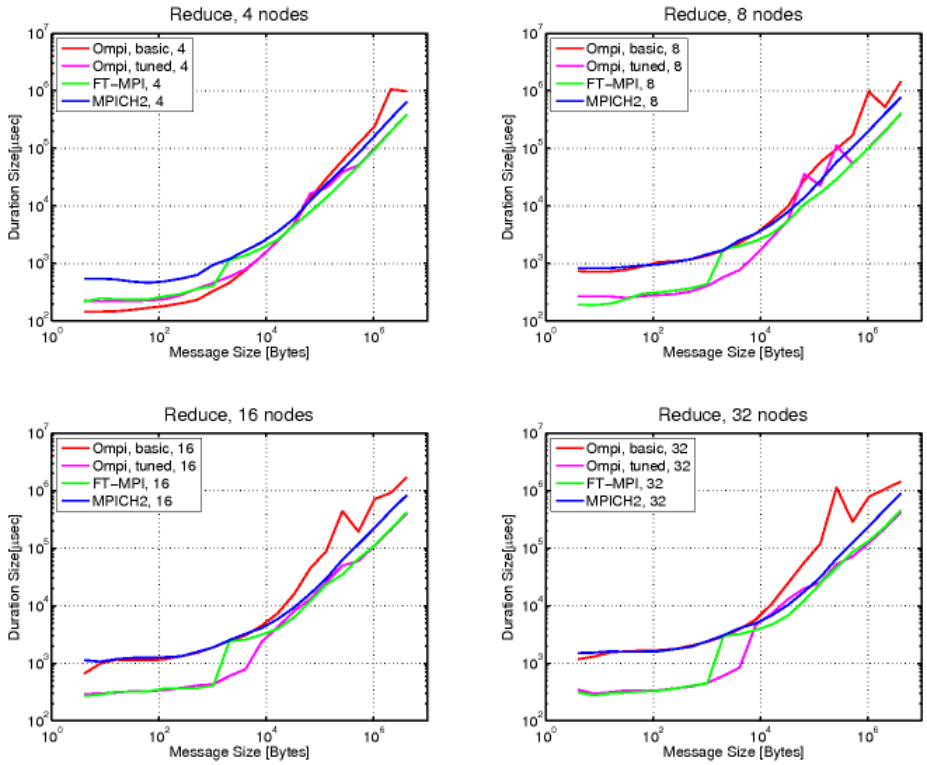
11 Appendix B

The below figures illustrate the absolute and relative performance of FT-MPI verses both the default Open MPI collectives module, the UTK developed Tuned collectives module and the current MPICH-2 implementation.



11 Appendix B continued.

Performance of FT-MPI compared to Open MPI default, UTK developed tuned module and MPICH-2 for a MPI Reduce operation.



11 Appendix C.

Additional Derived data-type performance of FT-MPI compared to MPICH-2 and LAM/MPI.