FT-MPI, Fault-Tolerant Metacomputing and Generic Name Services : a Case Study

David Dewolfs, Jan Broeckhove, Vaidy Sunderam, Graham E. Fagg

Depts. of Math and Computer Science of the University of Antwerp (Antwerp, Belgium), Emory University (Atlanta, GA, USA) and the University of Tennessee (Knoxville, TN, USA)
{David.Dewolfs,Jan.Broeckhove}@ua.ac.be, vss@mathcs.emory.edu, fagg@cs.utk.edu

Abstract. *There is a growing interest in deploying MPI over very large numbers of heterogenous, geographically distributed resources. FT-MPI provides the fault-tolerance necessary at this scale, but presents some issues when crossing multiple administrative domains. Using the H2O metacomputing framework, we add cross-administrative domain interoperability and "pluggability" to FT-MPI. The latter feature allows us, using proxies, to transparently replace one vulnerable module - its name service - with fault-tolerant replacements. We present an algorithm for improving performance of operations over the proxies. We evaluate its performance in a comparison using the original name service, OpenLDAP and current Emory research project HDNS.*

Keywords: FT-MPI, H2O, metacomputing, fault-tolerance, heterogeneity

1 Introduction

Over the course of the last ten years, clusters running some implementation of MPI have become some of the most popular supercomputing platforms. Recently, there has been a growing interest in clustering resources that feature extensive geographical distribution across multiple Administrative Domains (ADs). This raises the issue of fault-tolerance. FT-MPI [7] differs from other solutions to the fault-tolerance problem [3,4,6,10,5], in that it allows the application itself to restore it's own state, instead of relying on automated - but potentially unscalable - solutions like global distributed checkpointing. This makes it an interesting solution for highly geographically distributed, heterogenous resources with a need for customized, lightweight recovery mechanisms.

However, FT-MPI is currently confined to single ADs. Also, bottlenecks and potential single points of failure (SPoFs) become an issue when deploying it over slower AD interconnects. One of the critical modules is the FT-MPI name service (NS). We have previously addressed these points [2,1] by developing a proxy-based solution which allows FT-MPI administrators to use any NS of their own choice (including any fault tolerance features available with it). Further, we use features of the H2O metacomputing framework [8] to span multiple ADs without the need for individual accounts on each system.

In this paper, we focus on improving performance of operations over the proxies. We demonstrate the ability of our approach to transparently and scalably switch between different NSs. We will also present performance test data for the improved algorithms using different backend NSs.

2 Design Overview

2.1 Basic FT-MPI architecture

A running FT-MPI virtual machine (VM) deploys one FT-MPI runtime per node and a number of daemons to assist it in setting up and managing jobs: a *startup-daemon* on each node (semi-critical), one or more *notifier daemons* (non critical), and a single *naming daemon* (figure 1).

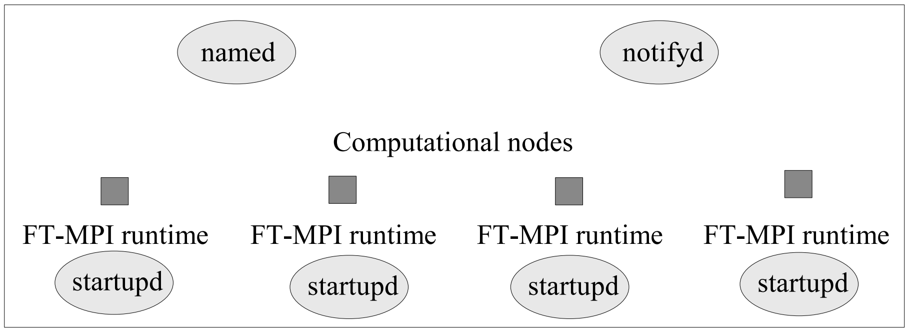

Fig. 1. a typical running FT-MPI system

Each VM needs exactly one naming daemon (however, a single NS instance can manage multiple VMs). It provides a custom NS and serves a crucial role in VM buildup, job startup and job recovery. Specifically, the FT-MPI runtime uses it to keep records on VM and job membership. To ensure data consistency, editing of records for job and task state in the NS is done by the FT-MPI runtime of single *leader node*. The leader edits these records during the error recovery phase to clean up job and task state. FT-MPI runtimes on other nodes are then notified of the changes through a system of callbacks. Leaders are *elected* through a custom call in the NS.

We note the following issues with the daemon in the currently available version of FT-MPI: 1) it constitutes a potential SPoF, as it is highly state-retaining and critically important for the general functioning of the VM, 2) it is also a possible choke-point when communicating over slow AD interconnects (this issue was recently addressed [13] and an adapted recovery algorithm should be added to future releases of FT-MPI) and 3) it does not support features like replication and load balancing, which would be desirable to improve scalability at very large VM sizes. We note that many generic name servers currently available do offer these features.

2.2 Extensions to the FT-MPI architecture

We use proxies to bridge between the custom FT-MPI NS protocol an any generic NS, enable an operator of an FT-MPI VM to use a NS of his own choice :

- instead of directly contacting the NS, components of FT-MPI contact a proxy which resides on the gateway between the single AD and the "outside world"; this proxy acts as a "*front-end*" to the real NS, translating FT-MPI protocol calls to a format that is understood by the real, "*back-end*" name service; the front-end does not retain internal state - thus, failures can be handled through simple measures like a trivial replication scheme or a restart
- all nodes on a single AD retain an open connection to the NS front-end for that AD, and each NS front-end retains a single connection with the NS back-end (hierarchical message forwarding)
- the NS front-end is implemented as a H2O "pluglet" making it fully remotely deployable by operators on any machine that runs an H2O kernel

The setup is best illustrated by the example in figure 2.

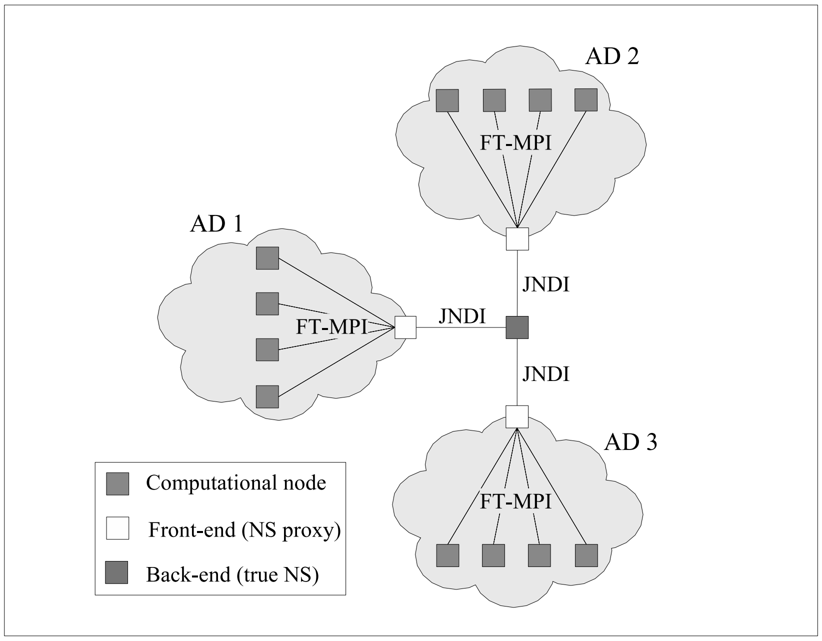

Fig. 2. An FT-MPI VM using proxies and a generic back-end NS

This approach allows FT-MPI to use one of a wide range of "off the shelf" NSs available. Many of these provide important fault-tolerance and performance features lacking from the current FT-MPI NS (load distribution, replication, checkpointing etc.).

The proxies are implemented in Java. This allowed us to use JNDI, the Java Naming and Directory Interface, which provides uniform access to a diverse set of NSs, ranging from LDAP to DNS. Any provider can make a NS "JNDI-enabled" by implementing a Service Provider Interface (SPI). All interaction with the NS is fully transparent to the user. Thus, using JNDI allows us to make access to the backend generic w.r.t. different NSs.

2.3 Concurrency, atomicity and JNDI

FT-MPI assumes a centralized, single-threaded NS which queues all incoming requests on receive. A number of its calls resolve compound operations like increment, compare and set etc. in a single atomic call. However, 1) the new design we propose has front-ends running in parallel and accessing the back-end concurrently and 2) certain single (atomic) calls in the NS have to be resolved through multiple primitives in JNDI, requiring separate lookups and subsequent binds. This introduces the possibility for concurrency problems, e.g. race conditions.

JNDI primitives We previously discussed a solution through the use of remote unreliable locks, composed from basic JNDI primitives[1]. We will show that it is possible to handle a majority of NS interactions without the use of said locks by exploiting the NSs single-update / multiple-callback architecture. To accomplish our goals, JNDI provides us with the following (relevant) atomic primitives:

- bind(*name,object*): binds *object*, which can contain an arbitrary number of fields to *name*; returns success or failure; appropriate exception is thrown if *name* is already bound
- rebind(*name,object*): replaces the current object bound to *name* by *object*, or acts identical to bind in case *name* hasn't been bound yet; returns success or failure
- lookup(*name*): returns the object bound to *name*; an appropriate exception is thrown in case of problems

JNDI also supports a callback mechanism, enabling us to register and "listen" for updates to the NS, very much like the original FT-MPI NS.

Leader election The most important part of custom functionality to implement is the leader election system. Once a leader gets elected, all editing of records for job and task state is done through him, eliminating the problem of concurrency. The FT-MPI NS implements leader election as a "grab the token" type of contest. The NS provides a custom call of the general form *swap(token,old_leader,contender)* which swaps the ownership of *token* from *old_leader* to *contender* if the current owner of *token* is *old_leader*. In other words: the first contender node to get its message handled by the NS gets to swap ownership of the token (and become leader) whilst a failure message is returned to the others on all subsequent messages.

An adapted election algorithm Given the primitives available to us, we perform leader election by implementing "grab a token" as "bind a token". For each token which is swapped during the lifetime of the VM, an object is stored in the NS with an *election_ count* keeping track of how many swaps have already been performed on it. This token is read during the initialization phase of the proxy and the counter is locally cached for later use. When an election takes place, all contender nodes send the appropriate message to their respective proxies and the following sequence of actions is performed:

1. the proxies each increase the cached leader counter for *token* by one, once - for all contenders who share the same proxy, the contest is resolved locally at the proxy
2. each proxy, for its respective local winner, attempts to bind an object under the name "<*token*>_<counter>" - the node for which the bind succeeds is the *winner node*, all others are *loser nodes*
3. the proxy acting for the winner node rebinds *token* with the new ownership data (triggering a callback) - the winner token becomes the leader and the outcome is relayed back to the new leader node - meanwhile, the proxies handling the calls for the respective loser nodes wait for a callback on a rebind for *toke*n, eventually relaying the outcome to their respective loser nodes as normal
4. if something goes wrong during the winner' actions in step 3 (non-atomic), this means something is wrong with the proxy, the gateway on which it resides, or its network connection; all of these will get nodes in their respective AD into trouble and register with the FT-MPI runtime as an error - the FT-MPI runtime will then recommence the recovery procedure (including a potential new leader election) as normal

This leaves us only with the problem of compound operations: what if something goes wrong with the leader in the middle of a compound operation? JNDI only allows for atomic operations on a single object at a time. This would lead to inconsistencies in the backend. We deal with this problem by using a single state object which contains pointers to all objects involved in the compound operation. We do not directly write to the objects themselves, but to a copy, keeping the old state intact until all actions in the compound operation have been performed. When ready, a rebind of the index record turns everything over to the new state within a single operation. This may leave spurious objects in the NS, but these can easily be cleaned up by an independent garbage removal process.

Results Advantages of this approach are 1) the ability to drastically reduce dependence on remote locks, enhancing performance by reducing the amount of JNDI calls that would normally be needed, and 2) the ability to do partial local resolution of the leader election process at the proxy, bringing down the amount of effective calls going out to the back end NS. This reduces the potential for choke-points on connections between different ADs and helps spreading load for very large, geographically dispersed VMs. Also, the number of callbacks is similarly reduced to one per proxy instead of one per node.

3 Evaluation

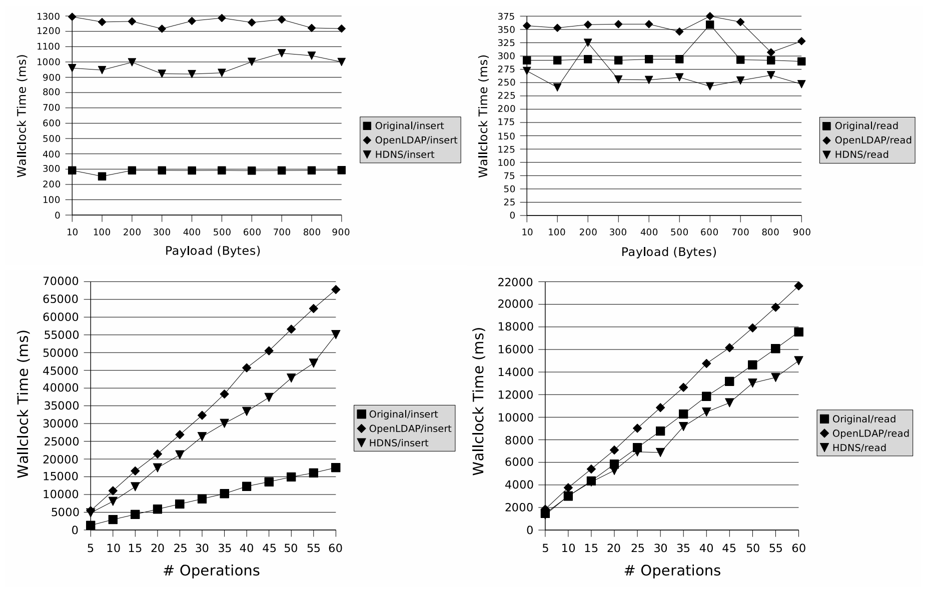

Fig. 3. Evolution of wall-clock time with increasing payload and # operations(read/write)

Setup and Experiments To demonstrate the ability of our setup to transparently switch between multiple NS backends,we performed a comparative experiment on two nodes: one in Atlanta (Georgia), USA, the other situated in Antwerp, Belgium. The node in Atlanta is a 4 CPU 2.8 GHz Pentium 4 with 1GB memory running Mandriva Linux 2006. The node in Antwerp is a 1.90GHz Pentium 4 with 256MB memory running Suse Linux 7. This setup was used in order to simulate the conditions which the design is aimed at: geographically distributed, heterogenous resources. The node in Atlanta ran the original FT-MPI NS, OpenLDAP or HDNS depending on the test case. The node in Antwerp ran a basic client program in both cases, plus the front-end in the case of the new design.

We ran a number of performance tests comparing the original NS with two alternatives: the LDAP-based OpenLDAP using the Berkeley DB, and HDNS [12]. HDNS is a naming service initially developed for the Harness Project[11]. While developing the SPI, a completely new version of HDNS has been designed and implemented. Both of the NSs tested support distribution and a number of features like fault-tolerance and persistency, which are not available in the original FT-MPI NS.

The following experiments were performed to evaluate scalability in terms of transaction size and frequency: 1) insert and read back entries with progressively growing payloads (10-900 B, using 100 B steps from 100 to 900 B) and 2) insert and read batches with a progressively growing number of equal-sized entries into the NS - measure wall-clock time for both cases. Ultimately, we want the new NS to be as scalable and stable as the original. We tested the performance of insertion and deletion without locks, allowed by the leader election mechanism described above.

Results We note that all experiments successfully ran to conclusion and left the back-end in a consistent state. From a practical point of view, we noticed that changing between OpenLDAP and HDNS was very easily accomplished. None of these experiments required any kind of code change or recompile of either the original FT-MPI code, or the Java-code for the proxies. A few changes to a configuration file and command-line parameters suffice to change NS back-ends from one experiment to another.

Looking at the figures, we conclude that the ability to do insertion without locking (though still less efficient than the original NS) provides us with a notable performance improvement over previous experiments in which we did use locking [1], the performance gain consistently being around 40%. It should prove interesting to do further research on improving performance of compound insertion operations, bringing figures even closer to those of the original NS. We also note that HDNS performs rather well as a backbone, outperforming OpenLDAP on both insert and read operations in both experiments. On read operations it even succeeds at slightly outperforming the original NS. We are currently investigating possible reasons for this remarkable behavior. Further, both graphs show linear growth on both insert and read for both experiments, proving that our design remains scalable and stable.

4 Conclusions

In this paper, we have discussed issues concerning the deployment of FT-MPI for large scale computations on highly geographically distributed, heterogenous resources. We have shown that "vanilla" FT-MPI poses some limitations in this area due to the nature of its naming service. We have presented a design, leveraging JNDI, which address these issues by enabling operators of an FT-MPI setup to "plug in" their own name services. This feature is highly desirable as existing "off the shelf" name services often do provide numerous features for improved fault tolerance and performance.

We have discussed an algorithm which allows us to implement a leader election system without locking, and note that it is possible to minimize the amount of locking in general. This results in a significant performance gain over previous implementations, both in terms of the amount of JNDI primitives needed and the amount of data transferred over connections between multiple administrative domains. We have presented experimental results which 1) confirm the efficacy of this approach, as well as 2) show the effective ability to transparently change between different back-ends, as demonstrated by our use of both LDAP and HDNS back-ends without significant changes.

References

1. D. Dewolfs, D. Kurzyniec, V. Sunderam, J. Broeckhove, T. Dhaene, G. E. Fagg. Applicability of Generic Naming Services and Fault Tolerant Metacomputing with FT-MPI. In *Proceedings of the 12th European Parallel Virtual Machine and Message Passing Interface - Euro PVM/MPI (Springer-Verlag Berlin LNCS 3666),* Sorrento (Naples), Italy, 2005
2. D. Kurzyniec and V. Sunderam. Combining FT-MPI with H20: Fault-tolerant MPI across administrative boundaries. In *Proceedings of the HCW 2005-14th Heterogeneous Computing Workshop,* 2005
3. A. Agbaria, R. Friedman. Starfish: Fault-tolerant dynamic MPI programs on clusters of workstations. In *Eighth IEEE International Symposium on High Performance Distributed Computing*, 1999, pp. 31
4. A. Bouteiller, F. Cappello, T. Herault, G. Krawezik, P. Lemarinier and F. Magniette. MPICH-V2: a fault tolerant MPI for volatile nodes based on pessimistic sender based message logging. In *ACM/IEEE SC2003 Conference*, 2003, pp. 25
5. Y. Chen, K. Li, J.S. Plank. CLIP: A checkpointing tool for message-passing parallel programs. 1997. Available at http://citeseer,ist.psu.edu/chen97clip.html
6. E. Elnozahy and W. Zwaenepoel. Manetho: Transparent rollback-recovery with low overhead, limited rollback and fast output. In *IEEE Transactions on Computers, Special Issue on Fault-Tolerant Computing*, 41(5), May 1992, pp.526-531
7. G. Fagg, E. Gabriel, Z. Chen, T. Angskun, G. Bosilca, J. Pjesivac-Grbovic and J. Dongarra. Process fault-tolerance: Sematics, design and applications for high-performance computing. In *International Journal for High Performance Applications and Supercomputing*. 2004.
8. D. Kurzyniec, T. Wrzosek, D. Drzewiecki and V. Sunderam. Towards self-organising distributed computing frameworks: The H2O approach. In *Parallel Processing Letters*, 13(2), 2003, pp. 273-290
9. S. Louca, N. Neophytou, A. Lachanas and P. Eviripidou. MPI-FT: Portable fault-tolerance scheme for MPI. In *Parallel Processing Letters*, 10(4), 2000, pp. 371-382.
10. G. Stellner. CoCheck: Checkpointing and process migration for MPI. In *10th International Parallel Processing Symposium*, 1996, pp. 526-531
11. M. Migliardi and V. Sunderam. The Harness Metacomputing Framework. In *The Ninth SIAM Conference on Parallel Processing for Scientific Computing, S. Antonio*, 1999
12. D. Gorissen, P. Wendykier, D. Kurzyniec and V. Sunderam. Integrating Heterogeneous Information Services Using JNDI. In *Proceedings of the HCW 2006 - 15th Heterogeneous Computing Workshop,* Rhodes Island, Greece, April 2006
13. G. E. Fagg, T. Angskun, G. Bosilca, J. Pjesivac-Grbovic, J. Dongarra. Scalable Fault Tolerant MPI: Extending the Recovery Algorithm. In *Proceedings of the 12th European Parallel Virtual Machine and Message Passing Interface - Euro PVM/MPI (Springer-Verlag Berlin LNCS 3666),* Sorrento (Naples), Italy, 2005, pp. 67