Computer Architecture Technical Report

TUC / RA-TR-2002-03

Date: 17th December 2002
(Last Build: 7th April 2003)

Feasibility of PACX–MPI for use in a Cluster–of–Clusters Environment

Mario Trams

Mario.Trams@informatik.tu-chemnitz.de

Chemnitz University of Technology

Department of Computer Science

Computer Architecture

Prof. Dr. W. Rehm

Feasibility of PACX–MPI for use in a Cluster–of–Clusters Environment

Mario Trams

`Mario.Trams@informatik.tu-chemnitz.de`

Abstract

There are several existing concepts and solutions that are intended for coupling several parallel computers in means of running a single MPI application on top of them. PACX–MPI [1] is one of them that relies on existing MPI libraries on the respective machines.
This paper discusses several facts and aspects for using PACX–MPI in an Cluster–of–Clusters (CoC) environment.

1 Introduction

There are several papers available from the PACX developers that provide a comprehensive view into internals and performance of PACX (some of them are [3], [4], [5], [6]). This paper is not intended to discuss internals of PACX in detail. Rather it focuses on the special implications for its use on cluster systems.

At the beginning, this papers gives a short overview on the use of PACX in a CoC environment that differs only marginally from the use in a cluster of MPPs. The second part describes various limitations of PACX that mostly appear when used in a CoC environment which is interesting for us. The third part describes some proposals how PACX can be changed and what can be added in order to improve it for better use in CoCs (while having no negative impact on the use with MPPs, of course). Finally, the last part suggests some concrete tasks for enhancements of PACX in the short–term future.

2 Basic PACX–MPI Operation

The principle operation of PACX–MPI (PArallel Computer eXtension; in the following text just called PACX) is quite easy. Figure 1 illustrates a setup with three different clusters or parallel computers in general.

In our case all three parallel computers or clusters are equipped with some kind of local communication network. As illustrated by figure 1, these networks can be completely different. At least one machine of each cluster needs to have a connection to the rest of the world. In most cases this will be a TCP connection but ATM is supported as well (version 4.1.4[1]). The “rest of the world” can be the whole Internet. A common intranet could do the job as well.

[1]everything described within this paper will assume PACX 4.1.4

All of the included parallel computers require a running MPI (MPI–1 compliant) that can be basically of any flavor. I.e. a proprietary MPI developed for this specific parallel computer or a more generic one — e.g. MPICH. Such an MPI is called *native MPI* as it is running natively on this machine.

It is a remarkable feature of PACX that it **only** requires an MPI as underlying system infrastructure. Neither PACX–specific daemons are required that run in background on some or all nodes, nor a multitasking operating system in that sense is required. All that is required are MPI libraries and the smallest program–unit PACX deals with is an MPI process. Hence, everything that is required besides application computing is derived from the pool of available MPI processes. Of course, this does not exclude the possibility to create new processes or threads when the underlying system does allow it.

2.1 Application Compilation

Most of the things described here are a short recap of [2].

Before applications can be compiled, PACX itself has to be compiled and linked against the native MPI. Building the PACX libraries is relatively easy. The following example builds the PACX libraries based on a MPICH-1.2.4 installation in `/home/ra/mtr/MPI/MPICH-1.2.4` and installs them in `/home/ra/mtr/WORK/PACX_MPICH-1.2.4/`.

```
# ./configure
--prefix=/home/ra/mtr/WORK/PACX_MPICH-1.2.4
--with-mpi-dir=/home/ra/mtr/MPI/MPICH-1.2.4
# make ; make install
```

There are no other options required, although there are some MPICH–specific configure options (refer to [2] for more details).

Of course, compilation of PACX has to be done for all machines with their respective native MPI involved in the meta computer later.

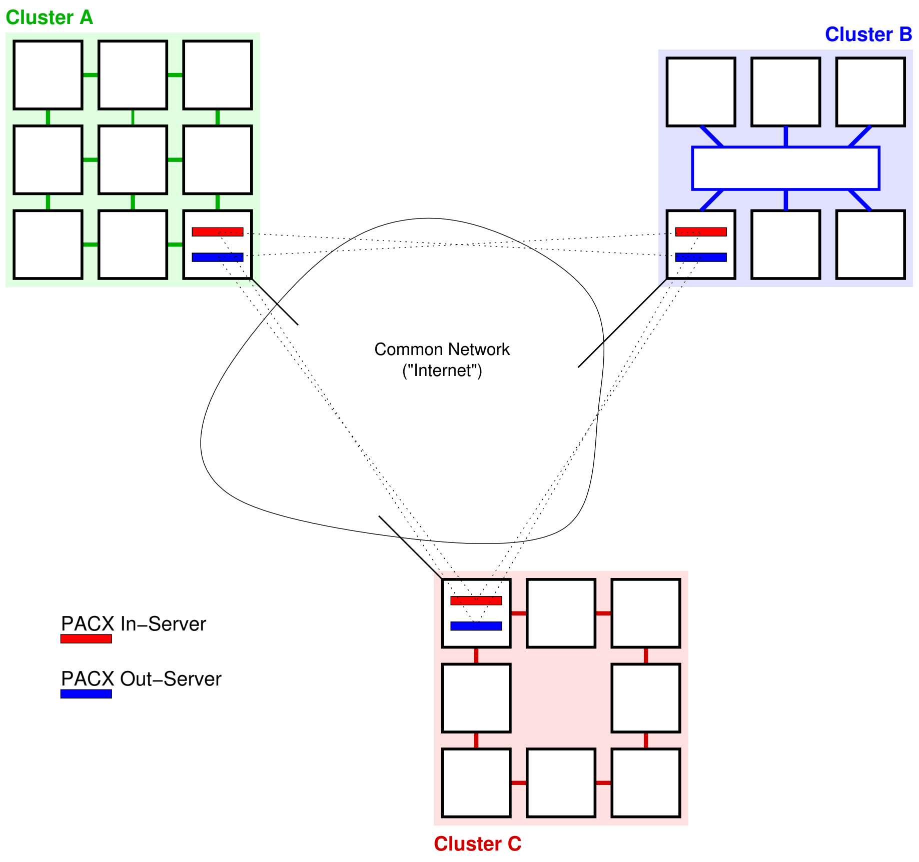

Fig. 1. Exemplary PACX Setup with three Clusters

Now, a standard MPI application can be compiled using the `pacxcc` script placed in
`/home/ra/mtr/WORK/PACX_MPICH-1.2.4/bin`:

```
# pacxcc app.c -o app
```

(Path has been omitted here.)

The application has to be compiled on all required architectures as well.

Note: In some cases it is required to extend the `pacxcc` script or a Makefile when used alternatively. Sometimes native MPIs require additional libraries to be linked besides the MPI library (so in case of SCI–MPICH where SMI and SISCI is required as well). However, adding additional library paths and libraries is rather easy.

2.2 Starting Applications

There's one basic thing for application startup: On each parallel computer the application has to be started with the startup mechanism provided by the native MPI of the respective machines. I.e. there is no `pacxrun` or something like that.

For parallel computers that feature a single Internet address for all nodes PACX offers a mechanism for an automatic startup. That is, the application just has to be started on one of the involved parallel computers. It will then automatically log into the remaining foreign machines to start up the application there.

Unfortunately, this is not possible in a conventional cluster as each node has a single IP address. So another mechanism has to be used instead that is working as follows:

- A so–called *Startup Server* has to be started on a "normal" computer known to all involved parallel computers. The startup server can be compiled independently from the actual PACX because it is a completely separate program and does not deal with native MPIs at all. An option for the startup server specifies the number of parallel computers involved in the meta computer.

  ```
  # startupserver 2
  ```

  configures the server for 2 parallel computers.

- When one part of the application has been started on a parallel machine, it parses `.hostfile`. This file is normally located in the directory of the application binary and contains information about the startup server and the rank of this parallel computer starting at 0. So if we consider two parallel computers, their `.hostfile` content could look like the following:
  `.hostfile` of first machine:
  ```
  Server jack 0
  ```

  `.hostfile` of second machine:
  ```
  Server jack 1
  ```

So the first cluster gets rank 0 in the "Meta–world" and the second rank 1. As the first cluster has rank 0 this means that all ranks in `MPI_COMM_WORLD` of user processes running on the first cluster will be smaller than those running on the second cluster.

- The startup server will wait until the number of expected machines have connected to it and will send some acknowledge back to each machine containing information about the single machines (number of MPI processes, addresses of special nodes, etc.).
- When all connections are established the MPI application can run as usual.

This startup process is, of course, completely transparent for the MPI application.

2.3 Inter–Cluster Communication

As the application has been linked against the PACX library, all calls to MPI routines are actually calls to PACX. When an MPI call is made by the application (say `MPI_Send()`, for instance), PACX can determine whether this is a local transaction on this cluster, or it requires communication with another one. In case of the local transaction PACX can pass the operation to the native MPI that sends it to the receiver located within the same cluster.

In case of a "global" transaction it has to be feed somehow to the respective node on the other cluster. Say, for instance, a message has to be sent from a node within cluster A to a node within cluster B (referring to figure 1). This is done by first forwarding the transaction via native MPI of A to that node in the cluster that has an `PACX_Out_Server` running. The `PACX_Out_Server` of A will feed the transaction via TCP/IP or ATM to a node of B that has an `PACX_In_Server` running. The `PACX_In_Server` of B can forward the message via the native MPI of B to its final destination.

As illustrated by figure 1, each cluster has one pair of in– and out–servers running. For the native MPI, these servers are treated as regular application processes. Each out–server has a connection to all in–servers and hence there is an overall N:N connectivity of the clusters (in figure 1 illustrated by dotted lines). It is not necessary to mention that all in– and out–servers have to reside in the same name space (i.e. direct access to the Internet). Although figure 1 shows both in– and out–servers in a single node, it is well possible to have them running on different nodes.

These nodes that have an in– or out–server running are also referred as *gateway nodes* in the following text. The remaining nodes that have the real application code running are also referred as *compute nodes*. The gateway nodes usually have no application code running.

2.4 A first Conclusion

First of all there's to say that PACX is definitively able to provide a sufficient framework for merging clusters consisting of different architectures right now. This is possible thanks to the all–over present TCP/IP.

3 Limitations of PACX–MPI

3.1 Inter–Cluster Communication Medium

As mentioned previously, TCP/IP or ATM are available for inter–cluster communication. While this (at least TCP/IP) is an absolute must for meta computing in sense of a worldwide coupling, it doesn't make use of the availability of high speed system area networks in the field of CoC systems.

Fortunately, this is not a real conceptual limitation as implementing alternative communication devices (Myrinet, SCI, InfiniBand, ...) shouldn't be that problem (see below).

3.2 Direct Inter–Cluster Communication

The concept of PACX requires a direct communication path between all in– and out–servers. This implies that the CoC framework must feature a **single** inter–cluster communication network.

This is because PACX lacks support of bridging between different inter–cluster networks. In other words, PACX requires a flat inter–cluster communication network without any additional routing hops.

A scenario which would require such a mechanism in a CoC–context is not very unrealistic. Just imagine two clusters A and B that might communicate via physical network 1. Now another cluster C needs to be integrated that is located a little bit far away from the other two clusters (say in another room). At least it is far enough located so that the use of network 1 is prevented because too long cables would be required. Another physical network 2 would have to be used instead. This network could, for instance, connect clusters A and C. Figure 2 illustrates this example.

So messages that are to be exchanged between clusters C and B have to make an additional "hop" over cluster A.

Exactly such structures are not supported by PACX yet. They could be realized when TCP/IP is being

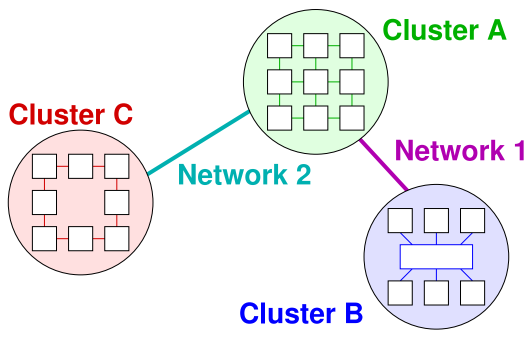

Fig. 2. Example with different Inter–Cluster Networks

emulated over different networks, but that is far from high–performance.

Note: It should be mentioned that PACX was never targeted for such configurations. Therefore it would be presumptuously to call this a weak point of PACX.

3.3 Startup Issues

As described previously, the startup server approach has to be used for starting the PACX framework on cluster systems. This is because PACX doesn't know the IP addresses of in– and out–servers prior to startup. Currently, PACX makes the following assignment: The MPI process with rank 0 of `MPI_COMM_WORLD` will become the out–server and the process with rank 1 the in–server (defined in `include/pacx.h`).

The problem is that the user does not have a direct influence on rank assignment. But in case of a CoC this is a crucial point as the servers have to be located on these nodes that maintain the fast interconnection to other clusters. Therefore it is absolutely required to have control over these decisions (at least indirectly, better directly).

Note: This is no problem as long as the whole CoC is covered by a single IP subnet and there is no special inter–cluster communication hardware but simple TCP/IP. In such a configuration each node can be a potential in– or out–server.

In case of MPICH, ranks seem to be numbered in the same order as they appear in the machine–file or `.pg`–file. So one can indirectly force the servers to be placed on desired nodes. However, it is not clear whether this can be generalized for all MPI implementations. Neither MPI–1.1 [10] nor MPI–2.0 [11] standards define a rule for initial rank assignment. Instead, MPI–1.1 states ([10], page 12, position 34):

> MPI does not provide mechanisms to specify the initial allocation of processes to an MPI computation and their binding to physical processors. It is expected that vendors will provide mechanisms to do so either at load time or at run time.

This doesn't explicitly speak about ranks, but they are included implicitly. MPI–2 does not put any requirements on the process startup either. So finally, each specific implementation of MPI may behave different in this point. And even more worse, some implementations might behave non–deterministic leaving the user no chance for any indirect rank assignments.

3.4 Difficulties with Batch Systems

Using PACX on a couple of parallel computers with independent batch systems is somewhat problematic. The only way using PACX in such a fashion would be the startup server mechanism where the startup server has to be started independently from the batch system(s) (e.g. on the users workstation). The automatic startup mechanism shortly described at the beginning of this paper does require an interactive access (remote shell) that is not working with batch systems. PBS [12], for instance, provides a mechanism for interactive jobs. Anyways, the actual problem is to get an interactive shell on all machines at the same time (see below).

In fact, a short investigation has shown that nobody seems to have used PACX in junction with batch systems.

Principally, there are no technical reasons that would speak against batch systems (we never tested it). However, there is an obvious machine utilization problem when all partitions of the PACX framework are not brought up within a short period. The actual application processing can begin once all participating partitions on various parallel computers have been started and all logical connections have been established.

So when one partition A comes to execution and another partition B is still waiting somewhere in a queue, A will block its machine without doing anything. And this problem is the same whether the batch system provides the capability for interactive jobs or not.

Note: Actually, this is a general problem not only dedicated to PACX.

4 How to cope with these Limitations?

4.1 Add more flavors of Inter–Cluster Communication

As previously mentioned, PACX could be expanded in a way to support just more kinds of communication networks for communication between clusters — in particular SANs. This should not have any conceptual impact on the PACX design at all. The sources of PACX are kept very modular in that point.

Therefore an integration of new media should be limited to the development of the communication routines and only minor changes on PACX itself.

Today, the most interesting candidates for new communication hardware seem to be Myrinet, SCI, and InfiniBand as it slowly evolves. Another interesting approach would be to use shared memory (i.e. standard SystemV shared memory) for communication. Figure 3 illustrates such a configuration.

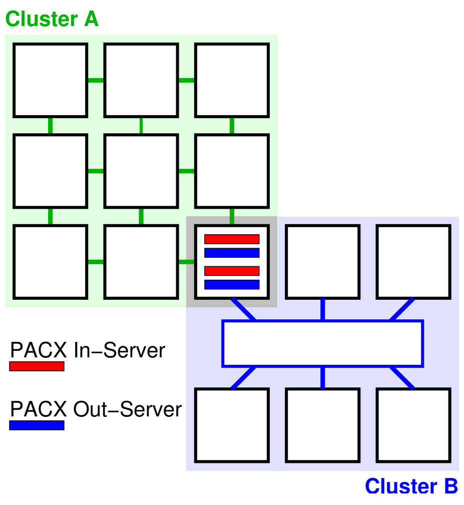

Fig. 3. Illustration of a Shared–Memory based coupling

There are two clusters, each equipped with different communication hardware. Both clusters have one node common that is equipped with both types of communication hardware. This node has running in– and out–servers for both clusters, and, of course, both native MPIs. Therefore messages that have to cross the cluster boundaries can be routed completely internally from out– to in–server via a shared memory protocol.

Note: One of the server pairs shown in figure 3 logically belongs to cluster A, and the other pair to cluster B!

Although this most likely leads to a bad scalability especially in view of technical/mechanical issues, it appears to be a nice and fast mechanism for coupling of two or three clusters.

4.2 Make use of abstract Inter–Cluster Communication

Besides adding capabilities for support of various technologies, it might be interesting to use a more generalized and abstract communication library. Of course, TCP/IP can already be considered as a method for communication with a high degree of abstraction. Messages can traverse an almost arbitrary number of almost any kind of hardware types while keeping this completely transparent for the using application. However, it is not any longer a secret that TCP/IP is not very well suited for use in system area networks due to high latency. Hence, the capabilities of the underlying physical media cannot be fully exploited.

But there are other projects that aim on abstract communication between nodes in a heterogeneous network environment. Two very ambitious projects in this area are Madeleine [7] and VMI [8]. The goals of both projects are similar: Developing a high–performance communication middleware that provides a virtualized/abstracted communication method on top of a bunch of real–existing system area networks.

Both Madeleine and VMI are intended as direct communication base for higher level libraries such as MPI. In fact, for both there is, respectively, a slightly modified version of MPICH available. Madeleine in its current version (III) is also capable of acting as gateway among different types of networks (refer to [7] for details). In case of VMI such functionality has been evaluated for VMI–1 [9], but it is not yet implemented in VMI–2.

The last paragraph describes that there already exists the capability of spanning a single MPI framework over a CoC system while making use of high speed system area networks. Hence, the adoption of PACX for that purpose might seem not necessary. However, using PACX for that purpose has the plus that it would be possible to use clusters with proprietary communication hardware and MPI libraries in a CoC context. Basically, this seems to be possible to achieve with VMI and Madeleine as well by using MPI as communication medium just as PACX does. But that is another story.

In case of using VMI or Madeleine in junction with the provided MPICH for running on a CoC, each node would have to run VMI or Madeleine functionality. In case of using one of these mechanisms just for inter–cluster communication in PACX, only the gateway nodes would be involved here (i.e. only the gateway nodes need to run VMI or Madeleine software).

The good thing with such a mechanism would be that PACX and in particular the in– and out–servers do not need to deal with things like message routing and line balancing as this is already provided by the middleware. Madeleine and VMI provide appropriate topology–description mechanisms as well. So add–ons and modifications of PACX itself should be held at a minimal level. The required efforts should be comparable to an integration of a special inter–cluster communication medium (e.g. Myrinet).

The only question that remains is whether such a design would be associated with a significant drop of performance or not (for whatever reasons).

4.3 Let PACX do the Routing

It is possible to let PACX do all the work by itself. That is, each subcluster of a CoC that has a connection to N external SANs would have to run N pairs of in– and out–servers.

Figure 4 illustrates an example of such a configuration.

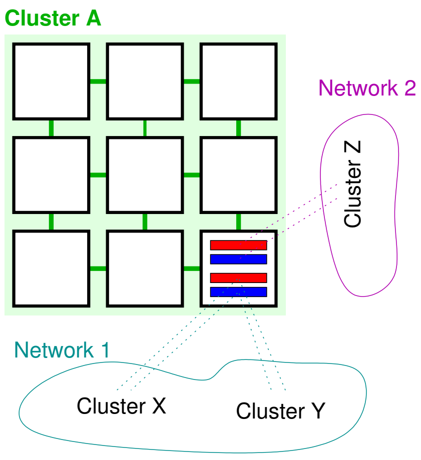

Fig. 4. CoC with different Inter–Cluster Networks in the same Node

In this example one Node of cluster A has two types of network adapters intended for communication with other clusters. For each network there's a separate pair of in– and out–servers.

When an in–server receives a message (say from network 1) and it is not intended for a compute node within this cluster, the in–server would forward the message to another out–server. This is nothing more than transaction routing on a different level than already performed by PACX (deciding whether a transaction has to be forwarded to an out–server on a gateway–node or to a regular compute node).

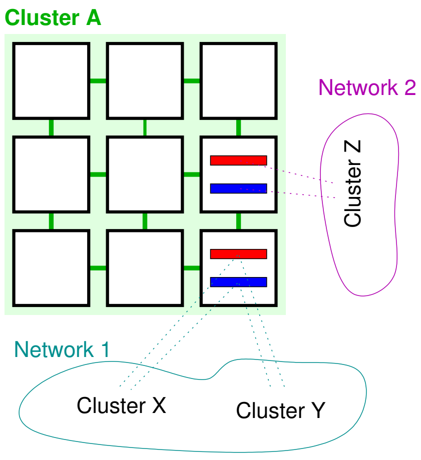

Fig. 5. CoC with different Inter–Cluster Networks in different Nodes

Figure 5 shows another possible scenario where two nodes in cluster A have different connections to other clusters. While in figure 4 shared memory could be used for transferring messages from an in–server to an out–server, the native MPI would have to be used in case of figure 5.

The exemplary configuration shown by figures 4 and 5 could be even more generalized as well. E.g.

- there could be more than two external networks for one cluster
- in– and out–servers of the same external network must not necessarily be located on the same node
- a transaction might need to cross multiple clusters before arriving at the final one
- etc.

So an in–server has to answer two questions when it has received some message that was identified as one not intended for a compute node within this cluster:

- To which out–server (residing within this cluster) the message needs to be forwarded?
- What about the connection to this out–server? (In which way the message has to be forwarded?)

For the latter one there are basically two options:

1. The out–server is located at another node on this cluster and the connection to this one is based on the local native MPI.
2. The out–server is located at the same node and messages can be forwarded via shared memory.

Note: A third option might come in mind where the out–server is reached neither via native MPI nor via shared memory. This is actually not a valid one and can be ruled out. The reason is that because by definition an in–server can forward messages only to a node inside the cluster it belongs to. A node that is reached neither via native MPI nor via shared memory cannot be part of the same cluster.

Besides these extended routing mechanisms in the server nodes the routing decisions of PACX running on regular compute nodes becomes more non–trivial as well. PACX has not just to decide whether a message has to be sent to another compute node or an out–server. In addition, it has to choose among several out–servers.

4.4 Modify the Startup Procedure

Regarding the startup and server assignment problem, the startup process of PACX could be slightly enhanced so that the user can directly specify on which nodes the in– and the out–server have to be located.

A direct but optional specification of the host name for in– and out–server (both separately) would provide much more flexibility. In these cases where the native MPI cannot be forced to assign ranks to processes on special nodes it would be a requirement. A definition of these nodes could be made inside one of the two existing PACX configuration files (`.hostfile` or `.netfile`), or in another file if something would speak against the integration into the existing files.

Note: Handling such assignments is not necessarily trivial as we have to consider SMP systems as well. That is, when a configuration file states that an in–server has to be located on node `foo`, only one process on node `foo` has to become in–server (same for out–server).

Another reason for a direct specification of the server nodes is to enable the automatic startup process as well for cluster systems. With a definition of the corresponding host names in a configuration file, PACX would have all the required information and there is a chance to bring it up on clusters without the annoying startup server.

4.5 The Batch System Problem

First of all it is to say that this problem is not that urgent in a CoC environment. It should be very likely that a single batch system can be used here that manages all clusters.

Nevertheless, caution must be taken here when the batch system treats individual partitions of a PACX framework as individual applications that are to be started on different clusters of the CoC. Actually, the batch system has to be aware of the dependencies and should bring all single partitions to execution at once.

Besides this, it would be a useful feature when the batch system picks up all the work required for partitioning the application. At least todays batch systems perform an automatic assignment of compute nodes. In addition to this, the following tasks could be done by such a system:

- Distribute the application transparently across several sub–clusters of the CoC.
  This includes the automatic generation of appropriate configuration files (not just for PACX, but for the native MPIs as well).
  Due to the heterogeneous nature of a CoC, the user should be able to request specific sub–clusters (for whatever reason; e.g. performance).
- Perform some kind of on–demand compilation of the application. This means that a user provides the source code instead of binaries and the batch system compiles them when needed.

It is not known to the author whether existing batch systems such as PBS [12] already offer functionality in these directions or how complicated it is to add such features. Therefore some further investigation and analysis has to be done here.

5 Suggestions — What to do next?

In order to get more confident with PACX, the implementation of a shared memory communication between out– and in–servers appears to be a good idea. The benefit of starting with shared memory instead of other mechanisms would be that it is a very "gentle" mechanism that does not bring things like unreliability into the game. When using SANs, one usually has to care about this special behavior and have to work around that bug etc. This makes the actual design more complicated than it appears in theory.

Once a shared memory bridging is working, other types of communication networks could be integrated as well.

However, the most interesting issue is to deal with the problem that PACX currently requires a flat inter–cluster communication network. Two different ways to work around this have been roughly discussed in this paper. Although it should have no major impact on the PACX core itself, an integration of the routing mechanisms into PACX seems to be a quite heavy task.

So in view of this particular problem interfacing PACX with one of the mentioned communication middleware frameworks appears to be the first choice. A more detailed analysis on how to implement such an interface will show whether there pop up some problems that are not seen at this time. Today, at least, there is no fundamental problem visible. Once it has been implemented, benchmarks will show the performance of such a system. For the case that there are major bottlenecks detected (for whatever reason), it should be considered to integrate such functionality more deeper in PACX as described herein. Perhaps major parts of the existing middleware could be reused for such an extension.

Enhancing the startup procedure of PACX by specifying host names for the servers as described would be a task that is somewhat independent from others. As discussed earlier, it is not necessarily required as long as there is a way for assigning ranks to nodes. A mechanism for explicit placement of servers would become required when PACX itself becomes responsible for routing and gatewaying between different clusters. Here not only the location of the servers has to be specified, but the type of connection between them as well. In other words, the whole topology of the CoC infrastructure has to be provided as input for the PACX system.

Having the automatic startup procedure in mind, the capability of a direct specification of sever nodes would be a valuable feature — even when the decision is made to go for a middleware approach regarding heterogeneous inter–cluster communication. When it turns out that a middleware approach is not a good choice (for whatever reason), it would be even more important to have such a mechanism ready so that it can be extended towards a complete topology description. So a development in this direction could be done in parallel to others.

6 Conclusions

This paper gave an overview on the implications of using PACX in a Cluster–of–Clusters environment. While PACX is basically capable to be used in such a context where it actually was not specifically designed for, there are some obstacles that were discussed herein.

Of course, possible solutions for these were given as well. In order to provide a short view of actions to do, here's a summarizing list of the author's suggestions:

- add more Inter–Cluster communication media; start with a Shared Memory connection
- let the PACX server nodes be running on top of a middleware layer for heterogeneous communication (e.g. VMI, Madeleine, or others)
- alternatively (or concurrently?) add more routing and gatewaying functionality to PACX
- implement a mechanism for better specification of server node location and application startup
- analyze the batch system issue in more detail

References

[1] PACX website:
`www.hlrs.de/organization/pds/projects/pacx-mpi/`

[2] PACX User Manual
`<PACX Source>/doc/pacx.ps`

[3] Thomas Beisel, Edgar Gabriel and Michael Resch: *An Extension to MPI for Distributed Computing on MPPs.*
In Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science. Springer, 1997.
`www.hlrs.de/people/resch/PAPER/PVMPI97/pvmmpi.ps`

[4] Edgar Gabriel, Michael Resch, Thomas Beisel and Rainer Keller: *Distributed Computing in a Heterogeneous Computing Environment.*
In Proc. 5th European PVM/MPI Users' Group Meeting, number 1497 in LNCS, pages 180– 187, Liverpool, UK, 1998.
`www.hlrs.de/people/gabriel/PAPER/liverpool98.ps`

[5] Michael Resch, Thomas Beisel, Holger Berger, Katrin Bidmon, Edgar Gabriel, Rainer Keller and Dirk Rantzau: *Clustering T3Es for Metacomputing Applications.*
In Cray User Group Proceedings, 1998.
`www.hlrs.de/people/resch/PAPER/CUG98/`

[6] Edgar Gabriel, Michael Resch and Roland Rühle: *Implementing MPI with Optimized Algorithms for Metacomputing.*
In Proceedings of the Third MPI Developer's and User's Conference. MPI Software Technology Press, 1999.
`www.hlrs.de/people/gabriel/PAPER/atlanta99.ps`

[7] Madeleine website:
`www.ens-lyon.fr/~mercierg/mpi.html`

[8] VMI website:
`vmi.ncsa.uiuc.edu`

[9] Sudha Krishnamurthy: *Design of a Gateway Protocol using VMI for Inter–Cluster Communication.*
Technical Report, NCSA and Department of Computer Science, University of Illinois, Dec. 1999.

[10] Message Passing Interface Forum: *MPI: A Message–Passing Interface Standard* (MPI–1.1), June 12, 1995.

[11] Message Passing Interface Forum: *MPI–2: Extensions to the Message–Passing Interface* (MPI–2.0), July 18, 1997.

[12] Portable Batch System website:
`www.openpbs.org`