Communication Target Selection for Replicated MPI Processes

Rakhi Anand, Edgar Gabriel and Jaspal Subhlok

Department of Computer Science, University of Houston,
{rakhi, gabriel, jaspal}@cs.uh.edu

Abstract. VolpexMPI is an MPI library designed for volunteer computing environments. In order to cope with the fundamental unreliability of these environments, VolpexMPI deploys two or more replicas of each MPI process. A receiver-driven communication scheme is employed to eliminate redundant message exchanges and sender based logging is employed to ensure seamless application progress with varying processor execution speeds and routine failures. In this model, to execute a receive operation, a decision has to be made as to which of the sending process replicas should be contacted first. Contacting the fastest replica appears to be the optimal local decision, but it can be globally non-optimal as it may slowdown the fastest replica. Further, identifying the fastest replica during execution is a challenge in itself. This paper evaluates various target selection algorithms to manage these trade-offs with the objective of minimizing the overall execution time. The algorithms are evaluated for the NAS Parallel Benchmarks utilizing heterogeneous network configurations, heterogeneous processor configurations and a combination of both.

1 Introduction

Idle desktops have been successfully used to run sequential and master-slave task parallel codes, most notably under Condor [1] and BOINC [2]. The distributed, heterogeneous and unreliable nature of these volunteer computing systems make the execution of parallel applications highly challenging. The nodes have varying compute, communication, and storage capacity and their availability can change frequently and without warning. Further, the nodes are connected with a shared network where available latency and available bandwidth can vary. Because of these properties, we refer to such nodes as *volatile* and parallel computing on volatile nodes is challenging.

The most popular approach for dealing with unreliable execution environments is to deploy a checkpoint-restart based mechanisms, as has been done (among others) by MPICH-V [3], OpenMPI [4] or RADIC-MPI [5]. A smaller number of projects are using replication based techniques, such as P2P-MPI [6] or rMPI [7]. We have recently introduced VolpexMPI [8], an MPI library which tackles the challenges mentioned above by deploying two or more copies of each MPI process, and utilizes a receiver based communication model with a sender

side message logging. The design of VolpexMPI avoids an exponential increase in the number of messages with increasing degree of replication by ensuring that exactly one physical message is transmitted to each receiving process for each logical receive operation. On each process, a priority list of the available replicas corresponding to every other communicating process is maintained. Clearly, the ordering of processes in this list will have a fundamental impact on the overall performance of the application. We refer to this problem throughout this paper as the *target selection* problem.

This paper evaluates five different approaches for target selection: a process team based approach as described in [8], a network performance based approach, an algorithm based on the virtual time stamps of the messages, a timeout based algorithm, and a hybrid approach deploying both network parameters as well as virtual time stamps. The algorithms are evaluated for the NAS Parallel Benchmarks for heterogeneous network configurations, heterogeneous processor configurations and a combination of both.

The remainder of the paper is organized as follows: section 2 gives a brief overview of VolpexMPI. Section 3 describes the five target-selection algorithms in detail. The algorithms are than evaluated for various hardware configurations in section 4. Finally, section 5 summarizes the results and presents the ongoing work.

2 VolpexMPI

VolpexMPI is an MPI library designed to deal with the heterogeneity, unreliability and the distributed nature of volunteer computing environments. The key features of VolpexMPI are:

1. *Controlled redundancy:* A process can be initiated as two (or more) replicas. The fundamental goal for the execution model is to have the application progress at the speed of the fastest replica of each process.
2. *Receiver based direct communication:* The communication framework supports direct node to node communication with a *pull* model: the sending processes buffer data objects locally and receiving processes contact one of the replicas of the sending process to get the data object.
3. *Distributed sender based logging:* Messages sent are implicitly logged at the sender and are available for delivery to process instances that are lagging due to slow execution or recreation from a checkpoint.

The receiver based communication scheme along with the distributed sender based message logging allows a receiver process to contact any of the existing replicas of the sender MPI rank to request a message. Since different replicas can be in different execution states, a message matching scheme is employed to identify which message is being requested by a receiver. For example, it is not sufficient for process zero to request a message with a particular tag on communicator `MPI_COMM_WORLD` from process one, if the application would send, over its lifetime, multiple messages with this particular signature. To distinguish between

the different incarnations of each message, VolpexMPI deploys a virtual timestamp to each message by counting the number of messages exchanged between pairs of processes. This virtual timestamp along with the tuple [communicator id, message tag, sender rank, receiver rank] uniquely identifies each message for the entire application execution. These timestamps are also used to monitor the progress of individual process replicas for resource management.

The overall sequence of operations for a point-to-point communication is as follows: Upon calling `MPI_Send`, the sending process only buffers the content of a message locally, along with the message envelope, which includes the virtual timestamp, in addition to the usual elements used for message matching. In case any of the replicas of the receiver process requests this particular message, the sender process will reply with the corresponding data.

The receiving process polls a potential sender and waits for the data item. Note, that there are two possibilities that have to be handled if a sender process does not currently have a matching message. First, the data might not be available yet, because the sender process lags the receiving process. This scenario is recognized by the sender by comparing the virtual time stamp of the request message with the most recent message having the same tuple [tag, communicator, sender rank, receiver rank]. In this case, the library will simply postpone the reply until the request message is available.

The second scenario is that the message is not available anymore, e.g., because the circular buffer used for sender side message logging has already overwritten the corresponding data item. This scenario only occurs, if the replica of the receiver process is lagging significantly behind the sender process. In this case the sender process will not be able to comply with the request. As of now we are not handling this situation. Thus, the lagging replica keeps on waiting for a particular message. However, there are various ways in which such a scenario can be handled, such as a time-out based mechanism, or an explicit reply indicating the inability to comply with the request. The long-term goal is to coordinate the size of the circular buffer with checkpoints of individual processes, which will allow guaranteed restarts with a bounded buffer size.

As different replicas of an MPI rank can be at significantly different stages of the execution, each process has to be able to prioritize the available replicas for each MPI rank. This is discussed in the next section.

3 Target Selection Algorithms

In order to meet the goal that the progress of an application correspond to the fastest replica for each process, the library has to provide an algorithm which allows a process to generate an order in which to contact the sender replicas. This is the main functionality provided by the target-selection module. The algorithm utilized by the target-selection module has to handle two seemingly contradicting goals: on one hand, it would be beneficial to contact the "fastest" replica from the performance perspective. On the other hand, the library does not want to slow-down the fastest replica by making it handle significantly larger number

of messages, especially when a message is available from another replica. The specific goal, therefore, is to determine a replica which is "close" to the execution state of the receiver process.

The team based approach, the original algorithm implemented in VolpexMPI [8], divides the processes into teams at startup, with one replica of each process in every team. Processes communicate within their own teams and contact a process from another team only in case of failure. Since processes are communicating exclusively within a team, fast processes are bound to communicate with slow processes, causing the application to execute at the speed of slow processes. A slow process can be defined as a process running on a slow internet connection, having a slow processor speed, or busy in some other work. Thus, in order to advance application at the speed of fast processes, there should be a mechanism where each process can select their communicating partner. However, team based approach does not provide any such mechanism. In order to solve this problem, different algorithms based on network performance, timeout, virtual timeout, and hybrid approach were developed and implemented.

Network Performance Based Target Selection Two key elements of network performance are latency and bandwidth. These parameters are used to establish an order among the replicas of an MPI rank on each process. For this, each process will use all of the known replicas of an MPI processes in a round-robin fashion for regularly occurring communication and time the operations. After receiving a fixed number of messages from each replica of the same rank, the receiver process calculates the latency and bandwidth corresponding to each sender process. Priority for the future is based on these parameters.

Note, that the algorithm has the ability to restart the evaluation process after a certain period of time, e.g. a certain number of messages to that process, or in case the estimated bandwidth value to the currently used replica changes significantly compared to the original evaluation. An important disadvantage of this approach is that, if one of the replicas used has an extremely slow network connectivity, the evaluation step will bring the fast running processes also to the speed of the slowest one for the duration of the evaluation.

Timeout Based Target Selection In this approach, each process waits for the reply from a replica for a limited time. If the requested data is not available within the predefined time frame, the process switches to another replica and requests the same message. A technical challenge is how to deal with the data of the original abandoned replicas, since the user level message buffer should not be overwritten by that process anymore. Thus, the library has to effectively cancel the original request before moving to the next replica. If the data from the first (slow) replica comes in, the data will be placed into the unexpected message queue of the library instead of the user buffer, and can safely be purged from there. For this, VolpexMPI maintains a list of items that need to be removed from the unexpected message queue(s).

One drawback for this algorithm is that it is difficult to define a reasonable value for the timeout. If the timeout value is too small, processes change their

target too frequently, whereas if the timeout value is large, processes will continue communication with slow processes for too long. Therefore, setting the correct threshold value plays a very important role. Another disadvantage is that all slow processes will try to contact fast processes making them handle more requests which may slow down the fast processes.

Virtual Time Stamp Based Target Selection This algorithm employs the virtual timestamps of messages to compare the state of sender and receiver processes. As explained in section 2, a virtual timestamp is the message number for communication between a pair of processes. Each sender process attaches its most recent timestamp for the same message type, i.e. message with same tuple [communicator id, message tag, sender rank, receiver rank] when replying to a receiver request. The receiver compares the timestamps from different senders to determine the execution state of the processes.

The overall approach starts by using the teams as created by the team-based algorithm. If the difference between the timestamps of two processes exceeds a certain threshold value, a process can decide to switch to another replica for that particular rank. Ultimately, it will choose the replica closest to its own execution state. Similarly to the timeout based approach, the major difficulty in this algorithm is to decide on good values for the threshold, i.e. when to switch to next replica. This threshold value must not be too small to avoid frequent change in targets, nor too large to avoid long detection time of slow targets.

Hybrid Target Selection This algorithm combines the network based algorithm and virtual timestamp based algorithm. Each process first sends a message to each replica and decides the preferred target based on the best network parameters. In order to determine the best target, pairwise communication is initiated during the initialization of the application. As a result each process is communicating with fastest communicating replica. This might make the fast running processes to slow down, since it has to serve potentially multiple instances of each MPI process. In a second step, the virtual timestamp based approach is used to separate slow replicas from faster ones, i.e., if a process is lagging far behind the sender process it changes its target to the slower one. The result is a reduced communication volume to faster processes. In the long term we envision the first step to be replaced by a more sophisticated process placement strategy of the `mpirun`, which can utilize proximity information obtained from the BOINC server and from internet metrics as presented in [9].

4 Performance Evaluation

This section describes the experiments with VolpexMPI library and the results obtained.The tests presented here have been executed on a regular dedicated cluster, in order to achieve reproducible results and understand characteristics of our algorithm. The cluster is composed of 29 compute nodes, 24 of them having a 2.2GHz dual-core AMD Opteron processors, and 5 nodes having two

2.2GHz quad-core AMD Opteron processors. Each node has 1GB main memory per core and network connected by 4xInfiniBand as well as a 48 port Linksys GE switch. For the subsequent analysis, only the Gigabit Ethernet interconnect has been utilized.

The NAS Parallel Benchmarks(NPBs) are executed for 8 process and the problem size B. For each target selection algorithm we record and compare the results obtained to the numbers obtained with the team based approach, the original target selection algorithm presented in [8]. Tests have been executed using two replicas per MPI process, henceforth denoted as double-redundancy, and three replicas per MPI process, also called triple redundancy runs. Note, that for triple redundancy runs only CG, EP and IS benchmarks have been executed. BT and SP would require 9 processes for that particular, which due to restrictions on the network configuration could not be executed. The triple redundancy tests of FT failed due to the memory requirement for each process.

4.1 Results for heterogeneous network configurations

In this set of experiments we explore, the cluster switch has been configured such that the link bandwidth to eight nodes has been decreased from Gigabit Ethernet(1Gb/s) to Fast Ethernet (100Mb/s), creating a heterogeneous network configuration. For the double redundancy tests, processes have been distributed such that no two replicas of the same MPI rank are on same network, i.e. if process 0,A is on Fast Ethernet then process 0,B is on Gigabit Ethernet. Furthermore, all teams contain processes that use the Gigabit Ethernet and the Fast Ethernet network. For the triple redundancy tests, one team of processes is being executed on a single 8-core node, creating a third hierarchy level with respect to the quality of communication. Similarly to the double redundancy tests, teams have been initiated such that each process has one replica on each of the three hierarchy levels, and all teams contain some processes from all three hierarchy levels. The challenge for the algorithms are to modify the original teams provided at startup such that (ideally) one set of replicas only contain processes on fast nodes, and the other one only consists of slow nodes.

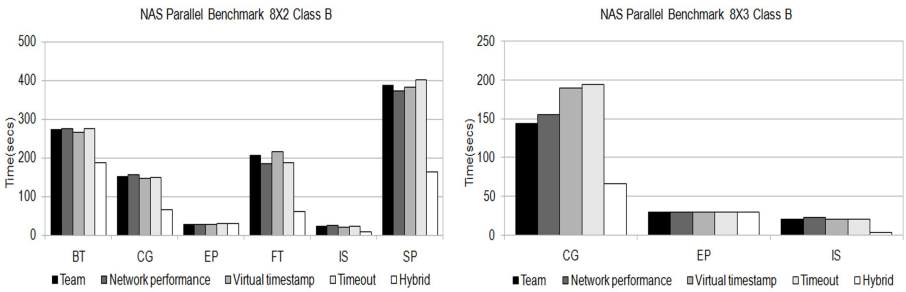

Fig. 1. Comparison of all target selection algorithms for heterogeneous networks.

Fig. 1 shows the results obtained with all implemented algorithms for double(x2)(left) and triple(x3)(right) redundancy. The results indicate, that the execution time for the network based algorithm is almost equal to the execution time of the team based algorithm. The main reason for this somewhat surprising behavior is that in some instances the algorithm identifies the wrong replicas as being the 'fastest' target. This happens if a fast process could not send response to the asking process because it is waiting itself for the result from a slow process, and which will falsify the measurements. Consider a simple example, where process 0,A and 1,B are running on fast nodes and processes 0,B and 1,A are running on slow nodes. In a situation when process 0,A sends a request for a message to 1,B, and 1,B is in turn waiting for a message from 0,B, its reply to process 1,A might be delayed. Note, that for artificial test case the algorithm worked as expected. However, for more realistic applications/benchmarks such as the ones used here, the MPI processes are too tightly coupled together in terms of send and receive operations, and the network performance based approach does not provide good performance results.

Next, we document the results for timeout based algorithm. The results for this algorithm are almost similar to the results obtained from network based algorithm for very similar reasons. As explained in section 3, the threshold value plays an important role in the overall performance of the algorithm. For the results presented here, we used a threshold value of 0.5 seconds for the double redundancy tests and 1.0 seconds for triple redundancy tests. We did perform experiments with various other threshold values, with lower values resulting in frequent switching of targets and higher threshold values preventing any process from switching to another replica.

Similarly to the other two algorithms, the virtual timestamp based algorithm does not show for the double and triple redundancy runs any difference to the team based approach, which is due to the fact that the synchronized communication patterns used in most of the NAS parallel benchmarks does not allow processes to 'drift apart'. Thus, the speed of fast processes is reduced due to the communication taking place with slow process and the application advances with the speed of slow processes. Note furthermore, that this algorithm in theory is designed to handle varying processor speeds and not necessarily varying network parameters for homogeneous processor configuration, and thus the result is not entirely unexpected.

Finally, the results obtained by using the hybrid approach in which processes are first grouped according to the network parameters and lagging processes are identified in a second step, is showing the best performance overall. The overall execution time is matching the performance that the same application would achieve when using Gigabit Ethernet network connections only. Thus, this algorithm is a significant improvement over the team-based approach utilized in [8]. Analyzing the results of this algorithm reveal, that the main difference comes from the fact, that the network parameters are not determined by timing the regularly occurring MPI messages of the application, but by introducing a pair-wise communication step that is executed in `MPI_Init`.

4.2 Results for heterogeneous processor configurations

In order to analyze the behavior of the algorithms for systems comprised of heterogeneous processor, the frequency of 9 nodes has been reduced to 1.1 GHz, while all other nodes are running at full frequency, i.e 2.2 GHz. All nodes are connected through Gigabit Ethernet, in order to eliminate network influences. Again we mixed the nodes running on slow frequency and nodes running on full frequency and compared the performances of different algorithms implemented.

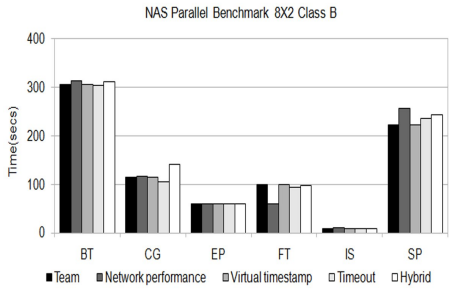

Fig. 2. Comparison of all algorithms for heterogeneous processor configurations.

Figure 2 shows the results obtained for this setting for all algorithms. The network based algorithm, timeout based algorithm and virtual timestamp based algorithm using double redundancy(left) runs are similar as discussed in the previous paragraph for similar reasons. In contrary to the previous section, the results obtained with the hybrid algorithm does not show any performance gain over other algorithms. This is due to the fact, that the network itself does not expose any hierarchies in this scenario, and therefore the pre-sorting of replicas and teams does not occur. In fact, nodes are grouped together without any proper order i.e each team consist of few processes running on slow nodes and other processes running on fast nodes.

4.3 Results for combinations of heterogeneous network and processor configurations

For the last set of experiments, the network connection to 8 of the 29 nodes has been once again reduced to Fast Ethernet. Furthermore, the frequency of the same 8 nodes has been reduced to 1.1 GHz while all other nodes are running at full speed. Similarly to the previous tests, teams have been initiated such that each process has one replica on a slow and on a fast node, and all teams contain some processes from all three hierarchy levels. For the triple redundancy runs, a third configuration consisting of 8 processes running at full frequency, but located on a single 8-core processor are interleaved with the other two sets.

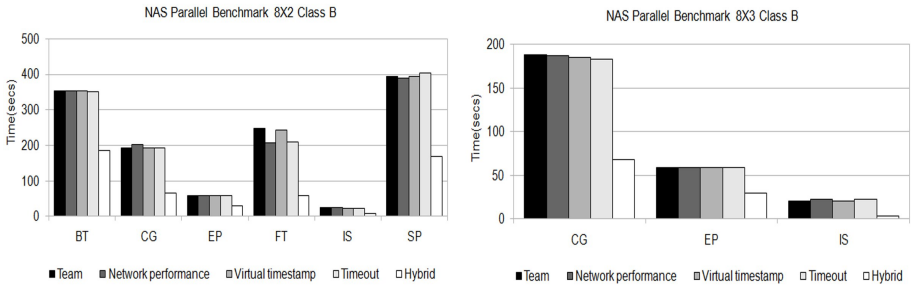

Fig. 3. Comparison of all algorithms for combination of heterogeneous network and processor configurations.

Figure 3 shows the results obtained for the double and triple redundancy runs. The results are similar to the results obtained in the previous sections, with the network based algorithm, timeout based algorithm and virtual timestamp based algorithm not being able to correctly identify the optimal configuration.

Table 1. Performance Results for redundancy 3 runs on different networks

	Volpex Team A	Volpex Team B	Volpex Team C
CG	87.93	67.69	184.29
IS	3.56	8.15	23.82
EP	29.69	29.72	117.83

The hybrid approach however gives the performance numbers similar to the results as if all processes are running on fast nodes. Also, for triple redundancy runs where all processes from each initial team are mixed, the hybrid algorithm is clearly able to group processes as if all three teams are executing on separate networks: Team A on shared memory, Team B on Gigabit Ethernet, and Team C on Fast Ethernet. This fact is highlighted by the results shown in table 1, which details the execution time observed by each individual team as identified by the hybrid target selection algorithm.

5 Summary

In this paper we presented and evaluated five different approaches for the target selection problem. The algorithms have been evaluated for the NAS Parallel Benchmarks for heterogeneous network configurations, heterogeneous processor configurations and a combination of both. The analysis reveals that the hybrid target selection algorithm shows a significant performance benefit over other algorithms for most (common) scenarios.

The ongoing work in this project includes a full evaluation of the new target selection method in a volunteer computing environment with a wider range of applications. On the algorithmic level we envision the initial placement to be driven by a more sophisticated process that can utilize proximity information obtained from the BOINC server and from internet metrics as presented in [9].

Acknowledgments. Partial support for this work was provided by the National Science Foundation's Computer Systems Research program under Award No. CNS-0834750. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

1. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: the condor experience. Concurrency - Practice and Experience **17**(2-4) (2005) 323–356
2. Anderson, D.: BOINC: A system for public-resource computing and storage. In: Fifth IEEE/ACM International Workshop on Grid Computing. (November 2004)
3. Bouteiller, A., Cappello, F., Herault, T., Krawezik, G., Lemarinier, P., Magniette, F.: MPICH-V2: a fault tolerant MPI for volatile nodes based on pessimistic sender based message logging. In: SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing, Washington, DC, USA, IEEE Computer Society (2003) 25
4. Hursey, J., Squyres, J.M., Mattox, T.I., Lumsdaine, A.: The design and implementation of checkpoint/restart process fault tolerance for Open MPI. In: Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE Computer Society (03 2007)
5. Duarte, A., Rexachs, D., Luque, E.: An Intelligent Management of Fault Tolerance in Cluster Using RADICMPI. In: B. Mohr, J.L. Träff, J. Worringen, J. Dongarra (Eds.) Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer, LNCS 4192 (2006) 150–157
6. Genaud, S., Rattanapoka, C.: Large-scale experiment of co-allocation strategies for peer-to-peer supercomputing in P2P-MPI. In: Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium. (2008) 1–8
7. Ferreira, K., Riesen, R., Oldfield, R., Stearly, J., Laros, J., Redretti, K., Kordenbrock, T., Brightwell, R.: Increasing fault resiliency in a message-passing environment. Technical report, Sandia National Laboratories (2009)
8. LeBlanc, T., Anand, R., Gabriel, E., Subhlok, J.: VolpexMPI: an MPI Library for Execution of Parallel Applications on Volatile Nodes. In: M. Ropo, J. Westerholm, J. Dongarra (Eds.) 'Recent Advances in Parallel Virtual Machine and Message Passing Interface', LNCS 5759, Espoo, Finnland (2009) 124–134
9. Xu, Q., Subhlok, J.: Automatic clustering of grid nodes. In: Proceedings of the 6th IEEE/ACM Workshop on Grid Computing, Seattle, WA (Nov 2005)