An Execution Environment for Robust Parallel Computing on Volunteer PC Grids

Hien Nguyen*, Eshwar Rohit*, Jaspal Subhlok*, Edgar Gabriel*,
Qian Wang†, Margaret S. Cheung†, David Anderson‡

* Department of Computer Science, University of Houston
† Department of Physics, University of Houston
‡ UC Berkeley Space Sciences Laboratory

***Abstract*—A pool of distributed volunteer PCs presents an extremely hostile environment for execution of communicating parallel codes due to system and network heterogeneity, varying availability, and frequent failures. Well known methods for fault tolerance, specifically replication and checkpointing, are challenging to deploy and not sufficient individually to provide continuous forward application progress. As the failure of a single logical process leads to application failure, the degree of redundancy needed for long running applications is too large to be practical. Checkpointing and rollback does not provide protection against slow and variable speed nodes and is impractical when system wide MTBF is in minutes or less, common for a moderate size volunteer computing pool. The approach taken in this research is to exploit both, but that presents formidable challenges; efficient checkpointing of distributed replicated processes, dynamic management of redundancy, quick restart in a distributed environment, and others. Proposed solution also leverages node selection based on availability prediction. The integrated runtime system is shown to effectively execute moderate size, coarse gain, communicating codes on a worldwide distributed volunteer environment, a new milestone in volunteer computing. The results provide new insight into how multiple techniques interact and contribute to robustness. The programming model is based on one-sided Put/Get calls to an abstract global shared space that works seamlessly with replicated processes. A Replica Exchange Molecular Dynamics code is employed to drive evaluation. The execution environment includes hosts on a University campus as well as hosts distributed around the world.**

***Keywords*-Volunteer computing, distributed computing, fault tolerance, BOINC, PC grids**

I. Introduction

Ordinary PCs have been employed successfully for large scale scientific computing, most commonly using Condor [30] or BOINC [5] as middleware. Condor, a scheduler that enables idle desktops to be employed for compute intensive applications, is deployed at 100s of sites worldwide. The BOINC middleware uses volunteered public PCs for scientific applications when idle. It has been remarkably successful, managing as much as 5 Petaflops of aggregate compute power and supporting over 60 scientific research projects. However, this is a tiny fraction of the volunteer compute power that could be exploited. The research presented in this paper aims to enhance the state-of-the-art for parallel computing on volunteer PC grids. We will refer to PCs made available for scientific computing when idle as *volunteer nodes (or hosts, or PCs)* and an execution environment composed of PCs connected by a LAN or Internet as a *volunteer PC grid (VPG)*; even if the PCs are managed hosts in an organization.

Volunteer PCs represent a potentially immense but *volatile* resource; they are heterogeneous in terms of architecture, networking, and operating system, prone to failure, and their availability to execute guest scientific applications can change suddenly and frequently based on the PC owner's actions. Execution of communicating parallel applications on volunteer PC grids is extremely challenging because process failures and slowdowns are frequent and the failure or slowdown of a single process impacts the entire application. The state of the art of parallel computing on volunteer nodes is generally limited to applications with "master-slave" or "bag-of-tasks" parallelism.

The context of this research is the VoLPEx (Parallel Execution on Volunteer nodes) project that has the goal of effective execution of parallel codes with coarse grain communication on volunteer nodes. The Volpex programming model is based on *autonomous redundant processes*, where a process can have multiple concurrent independent replicas that can be created at invocation, re-created from a checkpoint, or terminated, without coordination with other processes or replicas. The overall program progress is dictated by the instance of each process that is furthest ahead in execution. Volpex has developed two programming frameworks: the Dataspace API [19, 24] with a Put/Get communication model, and a subset of the MPI standard called the VolpexMPI [2, 3].

The central contribution of this paper is a runtime execution environment that achieves the key Volpex objective of continuous forward application progress despite network and host heterogeneity, routine failures, and other challenges of a volunteer environment. A suite of techniques is employed to make this possible: *node selection* to minimize the chance of failure during execution, *redundant processes* to provide failure resistance, *checkpointing* to allow application rollback, and *heartbeat monitoring* and *hot spares* for quick restarts. The runtime system innovations that allow application progress in such a harsh execution environment include: *i)* checkpointing and replication developed and implemented as a unified concept; typically minimal replication is deployed for seamless application progress on failure with checkpoint/restart to maintain a degree of redundancy, *ii)* checkpointing is co-operative;

more process replicas imply fewer checkpoints per replica *iii)* procedures to automatically "identify, kill, release and replace" healthy processes that are lagging in execution status; this is critical for managing heterogeneity and slowdowns that are not caused by failures, and *iv)* a "knob" to manage tradeoffs between resource expenditure and performance with control of the degree of replication, checkpointing frequency, and node selection standards. The goal of the integrated execution environment is to allow long running applications to execute effectively on volunteer PC grids.

The Volpex execution framework presented in this paper is in the context of the Dataspace programming framework. The execution environment leverages the Dataspace programming API, that normally stores application communication data objects, for storing checkpoints and other management data objects. The Volpex execution framework is integrated with the BOINC middleware that is used for the basic management of volunteer hosts. Some of the runtime features, in particular node selection, are implemented by modifying the BOINC framework. The challenging task of integration with BOINC was undertaken, first to avoid "reinventing the wheel", but equally importantly, to reach out to the substantial BOINC community of scientists as potential real-life customers for the software modules from this research.

The evaluation of this research is done on a volunteer environment that combines a group of hosts at the University of Houston with other hosts distributed worldwide. The evaluation is driven by a Replica Exchange Molecular Dynamics (REMD) code that is being used actively in physics research, e.g. [31]. The objectives of the evaluation are i) to demonstrate that Volpex execution framework can effectively manage distributed applications on volunteer hosts, and ii) to evaluate the role of node selection, checkpointing, and redundancy in enhancing the performance of long running codes on a volunteer PC grid.

While the primary goal of this research is to transform volunteer PC grids into "virtual clusters" to run a variety of parallel codes, the results from the research can potentially have wider significance. Other emerging platforms for distributed computing, in particular, computational clouds, also have to deal with heterogeneity, errors, and failures, especially as they scale up. The methods developed can play a role in improving reliability of clusters by employing unused cores to run redundant process replicas.

This paper is organized as follows. Section 2 discusses related work. Section 3 details the properties of volunteer nodes that are critical for the design of programming APIs as well as the execution environment. Section 4 introduces the concept of autonomous redundant processes. Section 5 discusses the Dataspace programming API as the context for this research. Section 6 discusses the design and implementation of the execution environment, including checkpointing, redundancy, and node selection. Section 7 presents evaluation results and Section 8 contains conclusions.

II. Related work

Idle desktops are widely used for parallel and distributed computing. The Berkeley Open Infrastructure for Network Computing (BOINC) [5] is a middleware system widely used for volunteer computing. Condor [30], is a workload management system that can effectively harness wasted CPU power. Other systems that build desktop computing grids include Entropia [21] and iShare [25]. Mechanisms applied for fault tolerance in PC grids, such as redundancy in BOINC and checkpointing in Condor, are important for long running sequential and bag-of-tasks codes, but are generally not sufficient for communicating parallel programs.

Linda [9], TSpaces [1], JavaSpaces [23] and many others represent a *Put/Get* model of coordination and communication among parallel processes based on a logically global associative memory. The Volpex Dataspace API [19, 24], the underlying framework for this research, is also based on *Put/Get* calls to a logically shared memory, the key enhancement being efficient support for distributed replicated processes. The notion of having a distributed shared memory programming model using an abstract data space has also been explored in [11], without however, support for fault tolerance.

Several implementations of the MPI specification have focused on fault-tolerance mechanisms. The vast majority of these projects rely on system level check-pointing and automatic roll back of the application, often relying on a third party system level library such as BLCR [13]. Representatives of these libraries include LAM/MPI [27], Open MPI [17], and MPICH-V [8]. Some approaches have utilized process replication to create robust MPI libraries, such as MPI/FT [7], P2P-MPI [16], rMPI [15] and VolpexMPI [3], the latter being developed as a complementary aspect of the work presented in this paper. P2P-MPI and VolpexMPI are the only libraries targeting volunteer computing within this group. Process replication is also deployed in the MOON framework [22] for MapReduce applications on volunteer resources.

The key innovation of this work is customizing and integrating replication and checkpointing to build an execution framework that can handle very high failure rates and other peculiarities of volunteer environments. A *Put/Get* model of communication is deployed as it is particularly suitable for a distributed environment. The runtime system will be adapted for an MPI implementation in the future.

An important component of the Volpex execution environment is a host selection module that leverages ample existing research. Multiple predictors of future availability of a host based on previous history are described and evaluated in [6, 26]. Prediction of application execution time in a volunteer environment is studied in [14, 29]. In contrast to research discussed above, this paper focuses on communicating parallel jobs rather than "bag-of-tasks" jobs, and on actual execution on a testbed rather than simulation. The predictor we employ is primarily based on recent host availability, a choice made partly because [6] showed that this simple predictor was nearly as effective as other complex and sophisticated predictors.

Finally checkpointing is an important component of the Volpex execution environment. Uncoordinated checkpointing [8] is the only viable option in a volunteer environment as coordinated checkpointing [27] requires finer synchronization and is difficult to handle with redundancy. Volpex currently employs application level checkpointing in the context of BOINC [4]. Our approach to sharing of checkpoints between replicas is partially motivated by [12].

III. Characteristics of Volunteer Hosts

We list the characteristics of volunteer computing hosts that present challenges for creating and running communicating parallel applications. In the subsequent sections, we address the design of the Volpex programming models and execution environment to address these challenges.

- *Heterogeneity:* Nodes are heterogeneous in terms of operating system, CPU model and speed, presence of a graphics processing unit (GPU), memory size, network bandwidth, and so on. They are on public internet implying that network connectivity to a server and to other nodes changes continuously.
- *Reliability:* Nodes are unreliable and untrustworthy. In addition to catastrophic failures, there can be Byzantine failures where incorrect results are returned. Some nodes have high error or failure rates because of hardware failures (due in some cases to overclocking). In principle, hackers can introduce malicious errors although this is extremely rare in practice.
- *Shared resource:* Most importantly, a volunteer node is not dedicated. Nodes may become unavailable because they are turned off, not allowed to compute due to user activity, or unable to communicate. Periods of unavailability may be short (e.g., when the user CPU usage exceeds a threshold and the guest process is disabled) or long (e.g., when an office machine is turned off for the weekend). A study of availability in BOINC systems [20] shows that nodes are available about 65% of the time, and about 35% of nodes are available and connected 99% or more of the time. The mean and median availability interval lengths are 20 hours and 8 hours respectively, and about 60% of the volunteer hosts have mean availability interval lengths greater than 4 hours
- *Accessibility:* Many volunteer hosts are behind firewalls, NATs, and HTTP proxies. Hence, only client initiated communication between a client and a Volpex/BOINC server works consistently. One implication is that scheduling must use a "pull" model in which the server waits for clients to request work. Also, direct communication between hosts may be available but cannot be taken for granted.
- *Resource abundance:* In volunteer computing environments, computational resources are often abundant and the primary challenge is to harness them effectively. In particular, it is generally reasonable to trade off reduced resource utilization to achieve reliable execution.

IV. Autonomous redundant processes

Our goal is an execution environment that converts a pool of volunteer hosts into a virtual cluster that supports reliable execution of parallel programs. Important considerations are redundancy and checkpointing with minimum coordination in the control layer. We refer to the approach as *autonomous redundant processes* outlined as follows.

- *Redundancy:* A process can have multiple concurrent executing replicas (or instances), typically two but it can be higher. Process replicas may be explicitly created at program invocation or a new instance may be created from a checkpoint to replace a slow or failed process instance, and/or to maintain a certain level of redundancy.
- *Autonomous processes:* Process replicas are not aware of the existence of, or coordinate with, other process replicas and largely act autonomously. A process can checkpoint its state without coordinating with other processes. A process can be recreated from a checkpoint irrespective of whether the original process is dead or alive, and without coordination with other replicas or other processes.

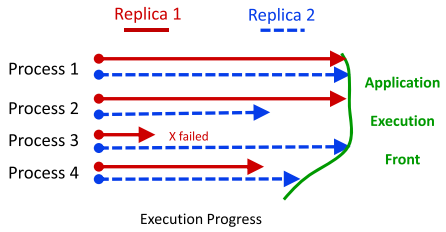


Fig. 1. Application progress is determined by the leading front created by the fastest replica for each process

The execution model ensures consistent and seamless application progress while at least one replica of each process is active; other replicas may be dead or lagging. This is illustrated in Figure 1. Lagging replicas may be be used for result verification and for providing protection from Byzantine faults due to software/hardware errors or malicious hosts. Redundant computations may appear "wasteful" but are important and complement checkpointing for a host of challenges; in particular, data corruption, malicious software, nodes and networks of varying speeds, transient delays, and frequent failures.

V. Background: Dataspace Communication API

The main contribution of this paper is an execution environment for reliable execution on volunteer hosts. The context of this research is the Volpex Dataspace programming framework. We summarize the Dataspace API and implementation here with more details available in [19, 24].

The Volpex Dataspace communication library consists of asynchronous calls to add, read and remove data objects to/from an abstract shared *Dataspace*, with each object identified by a unique *tag*, which is a global index into the Dataspace. The concept of a Dataspace is similar to that

of a tuplespace in Linda [9] and many variants. The main communication calls are abbreviated as follows:

Volpex_put(tag, data); A *Volpex_put* call writes the data object *data* into the abstract Dataspace identified with a *tag*. Any existing data object with the same tag is overwritten.

Volpex_read(tag); A *Volpex_read* call returns the data object that matches the *tag* in the Dataspace.

Volpex_get(tag); A *Volpex_get* call returns the data object that matches the *tag* in the Dataspace, and then removes that data object from the Dataspace.

Additional calls are available to retrieve the process Id, the number of processes, and for other housekeeping functions. The set of calls in the Dataspace API is minimal but sufficient to simulate basic message passing and shared memory style communication.

Dataspace Execution Model: The basic semantics of the communication operations are straightforward as listed with the API above. However, managing autonomous redundant processes, implying that multiple copies of the same process may execute asynchronously, is the main challenge. The key problem is that a logical call (with side effects) may be executed repeatedly at different times. To ensure consistency, the execution model consists of the following fundamental rules:

1) *Atomicity rule:* The basic *put/read/get* operations are atomic and executed in some global serial order.
2) *Single put rule:* When multiple replicas of a process issue a *Volpex_put*, the first writer accomplishes a successful operation. Subsequent corresponding *Volpex_put* operations are ignored.
3) *Identical get rule:* The first replica issuing a *Volpex_get* or a *Volpex_read* receives the value stored at the time in the Dataspace. Subsequently, replicas of the corresponding *Volpex_get* or *Volpex_read* receive the same value, independent of the time at which they are executed.

If these rules are followed, all process instances are guaranteed to execute identically. Application results are consistent for deterministic as well as nondeterministic parallel programs as long as the sequential execution of a single process code is deterministic.

Dataspace Implementation: A prototype implementation with a single Dataspace server process exists from prior research. The *atomicity rule* is satisfied with a single threaded server process. For *single put* and *identical get* rules of the execution model, the Dataspace server maintains two separate pools of storage:

- *Dataspace table:* Logically "current" data objects indexed with tags.
- *Log buffer:* Data objects that may not be current, but may be needed to process future *get* and *read* calls.

Each logical communication call is uniquely identified by the pair: *(process_id, request_number)*, where *request_number* is the current count in the sequence of requests from a process. This allows the server to discern new communication calls from subsequent replicated calls. If a communication call is recognized at the server as a new call, then normal operations are executed from the Dataspace table, while a replica call may be serviced from the Log buffer. The implementation is outlined in Figure 2.

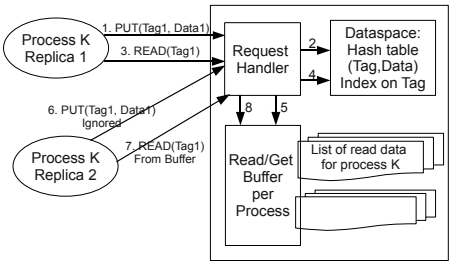


Fig. 2. Volpex Dataspace server design

In the current implementation, the Dataspace server is a single point of failure and a potential bottleneck. However, it should be noted that the server is normally hosted on reliable dedicated hardware. Distributed, multithreaded, and fault tolerant implementations are possible but a discussion is beyond the scope of this paper.

VI. Execution Environment

The execution environment is critical for parallel Volpex jobs to run effectively on volunteer hosts characterized by heterogeneity, fluctuating availability and failures, and limited accessibility, as discussed in section III. We first describe the basic functions and implementation of the execution framework, and then discuss the key features in more detail.

A. Execution manager overview

The basic tasks of the execution manager are as follows:

1) *Initiation:* A new Volpex job provides the execution manager with the number of worker processes and the list of application versions available (e.g. Windows, Linux, GPU types, etc.). Additional parameters include total FLOPs (reflecting nominal execution time), FLOPs between checkpoints, size of checkpoints, memory and disk requirements, communication bandwidth, etc. The execution manager starts recruiting hosts for the job by selectively accepting volunteer hosts that contact the server. A set of additional hosts are recruited as *hot spares*. When sufficient nodes are recruited, application execution starts simultaneously on all selected nodes; processes may have two or more replicas at initiation based on application request and execution manager policy.
2) *Progress:* During execution, the hosts create and store independent checkpoints. The execution manager monitors execution with heartbeats. When a replica dies or slows down so much that it becomes irrelevant, additional process replicas are created by activating hot spares, possibly from a checkpoint. Lagging live replicas are shut down.

Figure 3 outlines the implementation of the execution management framework. The BOINC middleware is used for core execution management tasks such as dispatching binaries and data files to hosts and verifying and collecting results. The host selection is implemented as an extension to the core BOINC scheduler. The Dataspace communication API is used by the hosts for sending heartbeats and checkpoints, and by the execution manager to activate or deactivate hosts. A shared database allows access to information across BOINC and Volpex Dataspace server boundaries.

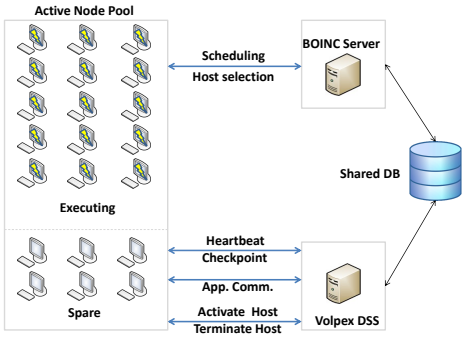


Fig. 3. System Execution Framework

B. Host selection

We first discuss the policy for host selection and then the mechanism for selecting hosts and initiating execution.

Policy: It is important to note that parallel applications typically make progress at the speed of the slowest process. The objective is to *maximize the minimum performance* rather than maximizing the average performance. We discuss the key considerations in the host selection policy.

- *Computation and storage capacity:* Only hosts above a minimum threshold of CPU capacity, memory storage, available communication bandwidth, and available disk capacity are selected. This reduces the probability that a single host becomes a bottleneck because of limited system or communication capacity.
- *Expected future host availability:* When a host fails or becomes unavailable, the execution may have to be restarted from the beginning, or from a previous checkpoint, while the rest of the hosts are blocked for communication. Considerable research has reported on the predictability of host availability in volunteer environments [6, 18, 26]. We have implemented a basic predictor called *Lastval* which is simply based on the availability of the host in the immediate past. This predictor, although very simple, has been shown to be competitive with the best predictors in simulations [6].

Mechanisms: Host selection has been implemented by modifying the basic BOINC scheduling policy. When an application request is obtained, the runtime system starts recruiting hosts as they contact the application server. Some additional hosts are recruited as *hot spares*. Note that volunteer computing with BOINC employs a "pull" based policy where hosts initiate all contact with the server. Hot spares are recruited ahead of time so that (i) execution does not block when another host is needed and the BOINC server has to wait until it is contacted by an appropriate host, and ii) to avoid the startup time of shipping executables etc. once a replacement host has to be activated. After a host is recruited, it is provided with the necessary binaries and data files, and then it contacts the execution manager, with a Dataspace API call, to request a process ID. Once sufficient hosts are recruited, execution is initiated on all hosts, by providing them with process IDs. Hot spares are denied a process ID, until they are needed. Depending on the user preferences and policy parameters, one, two, or more instances of each process may be initiated.

C. Checkpoint and Restart

Volpex design is based on integrated redundancy and checkpoint management. The challenges for checkpoint management in the Volpex execution environment stem from the following considerations: i) Volpex processes and process replicas are distributed and may be in different stages of execution, ii) there can be a varying number of process replicas, and iii) checkpointing overhead has to be managed even though an individual process that issues a checkpoint does not have the information necessary to make optimal checkpointing decisions. We discuss how these are addressed in the Volpex execution environment.

Currently only application level checkpointing is supported, although system level checkpointing (e.g. [13]) may be supported in the future. Uncoordinated checkpointing [8] is the only viable option in a volunteer environment as coordinated checkpointing [27] requires synchronization of all processes. Global synchronization to create a checkpoint is especially impractical when multiple replicas of a process exist and the objective is for the application execution front to move forward with the leading replica and to never have to wait for lagging replicas. Recovery from an uncoordinated checkpoint requires that re-executed communication calls from past logical execution states must also receive a logically correct response. However, this is a central feature of the Dataspace communication API designed to support redundancy and process restart from a checkpoint.

Volpex checkpoints are stored on the Dataspace server via a *Put(DataObject)* call. While the application issues the calls to create checkpoints, the execution manager controls the maximum frequency at which a checkpoint is generated and stored, in order to prevent the system from being overwhelmed by excessive checkpoints. Essentially, the execution manager ignores an application call to checkpoint if sufficient time has not elapsed since the previous checkpoint.

Recording of checkpointing is also co-operative between the replicas of a process. A new checkpoint is created and stored only when it represents meaningful application progress beyond the most recent recorded checkpoint across replicas. As replicated processes may be at different stages of execution,

it is inevitable that some process replicas will attempt to checkpoint their state after another replica has already recorded a checkpoint that represents a state further ahead in logical execution. A checkpoint that represents a state older than the state represented by the most recent checkpoint in the system is of no value. Such a checkpoint request is identified as an obsolete request with a logical timestamp mechanism and the request to checkpoint is ignored. An important benefit of such co-operative checkpointing is that the checkpoint frequency for a process instance decreases as the number of replicas increase. Also, lagging process replicas will not need to checkpoint at all.

Finally, a restart is achieved by activating a spare node with the most recent checkpoint of the process that needs to be restarted.

D. Runtime process control

Once execution is in progress, the main responsibility of the Volpex execution environment is to generate new process instances when needed and terminate process instances that are not useful for application progress. The default policy followed by the Volpex execution environment is to maintain the degree of redundancy with which an application was started. A new process instance is created when an existing process instance is *dead* or *obsolete*. We first discuss how dead and obsolete process instances are identified.

a) Dead processes: Every process periodically sends a *heartbeat* to the Dataspace server. If several heartbeats are missed, i.e., no heartbeat is received for a preset period of time, then the process is considered *dead*. Note that the reason may be that the host had a failure, the host process was terminated due to user activity, or there was a network outage.

b) Obsolete processes: A host may be providing heartbeats, but for a variety of reasons, it may be making very slow progress. As a result, it is possible that a process replica gets so far behind in execution that there is no benefit to the application in keeping it running. In this case, the process replica is considered obsolete. Obsolete processes are detected by a logical timestamp reported with every checkpoint request that is tracked by the execution manager.

Whenever a process instance is considered dead or obsolete, it is terminated if possible, and a spare host activated to take its place. A process instance is terminated by sending a *Kill* response to a heartbeat. Note that a dead process may become alive (a zombie) and send a heartbeat in the future, in which case it is terminated. The reason for terminating a process instance is that the host can be used for another application when healthy.

The creation of a process instance is simply done by providing the process ID to one of the spare nodes. After receiving a process ID, a new process automatically checks for the most recent checkpoint in the system for that process. If a checkpoint is found, execution starts from that state, else execution starts from the beginning.

VII. Usage and results

The Volpex Dataspace programming framework has been implemented, deployed and tested with several benchmarks and applications. The execution management system is integrated with the Volpex Dataspace system and the BOINC scheduler. Experimentation and validation was done on a volunteer host pool that consists of ordinary internet-connected desktops as well as compute clusters and lab computers. The results presented in this paper focus on validating the execution environment, rather than the overall usage of Volpex. The following are the specific objectives:

1) To demonstrate that the Volpex execution environment integrated with the Dataspace API and BOINC scheduler effectively converts a volunteer node pool to a virtual cluster capable of executing long running parallel jobs.
2) Evaluate the importance of the execution management mechanisms, specifically node selection, replication, and checkpointing, on the performance of parallel programs executed on a volunteer environment with extensive experimentation on a real-world testbed.

A. Testbed for experiments

For the results presented in this paper, the codes were executed on a volunteer environment managed by BOINC middleware consisting of clusters and desktops from UH Computer Science Labs, desktops from a UH Physics Lab, individual volunteer nodes on campus and a substantial number of volunteer nodes from around the world. The clusters and lab nodes inside UH are in active use for other tasks but available to BOINC/Volpex when idle. The most common configuration of a lab PC was a 1.86GHz Intel x86 processor with 2GB of memory. Computer science lab PCs were running Windows XP and most Physics Lab PCs were running Unix. Our scheduling policy allows only one Volpex process to run on a node at a given point in time. We verified that virtually all our experiments employed a mix of on-campus nodes and volunteer nodes from around the world. We also observed that on-campus and other nodes were of similar distribution in terms of execution quality; in particular the best and worst nodes in terms of execution quality were both a mix of on-campus nodes and other volunteer nodes.

The Dataspace server and the BOINC server were running on an AMD Athlon 2.4GHz dual core machine with 2GB memory running Ubuntu 9.01. The server and on-campus client nodes were on different subnets that are part of a 100Mbps LAN on UH campus.

B. Applications and benchmarks

Several programs have been developed for the Volpex environment including a *Sieve of Eratosthenes (SoE)* program, a *Parallel Sorting by Regular Sampling* program and a simple implementation of *Map-Reduce* framework. We have also adapted a *Replica Exchange Molecular Dynamics (REMD)* code from an active physics project to run under Volpex. For this paper, we report results only from the Sieve and REMD programs, so we describe them in more detail.

Sieve of Eratosthenes (SoE): is a well known algorithm for finding prime numbers. The Dataspace API was used to broadcast a block of new prime numbers to all processes in the parallel implementation.

Replica Exchange for Molecular Dynamics (REMD): is a real world application used in protein folding research [28]. Each node runs a piece of molecular simulation at a different temperature using the AMBER program [10]. At certain time steps, temperature data is exchanged between neighboring nodes based on the Metropolis criterion, in case a given parameter is less than or equal to zero. In our implementation of this code, the dataspace API is used for i) storing process-temperature mapping, ii) synchronization of the processes at the end of each step, iii) identification and retrieval of energy values from neighboring processes, and iv) swapping of temperatures between processes when needed.

In the REMD experiments we conducted, each temperature replica[1] represents a process which starts running simulations for a certain temperature. At the end of each step, neighboring temperatures may be exchanged between processes based on the Metropolis criterion. A snapshot of such exchange of temperatures is presented in Figure 4, from one of the experiments. It shows how temperatures are exchanged for an execution with 10 temperature replicas and 5 steps. In each step, temperatures that are swapped are highlighted in bold with the same background pattern.

# Step	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
1	290	300	310	320	**330**	**340**	350	360	370	380
2	290	**300**	**310**	320	340	330	360	350	380	370
3	**290**	**310**	300	340	320	360	330	350	380	370
4	310	**290**	**340**	300	360	320	350	330	370	380
5	310	340	290	360	300	350	320	370	330	380

Fig. 4. REMD Temperature(K) swaps: 10 temperature replicas for 5 steps

C. Experiments

The SoE and REMD benchmarks were employed for the experiments that are reported in this paper. All codes were executed with 32 processes. Only one process instance was placed on one host, meaning that a total of 32 , 64 or 96 hosts were employed for a degree of replication of 1, 2, and 3, respectively. The nodes available in the environment at a given time were up to 200. While the processes in both applications communicate regularly, the volume and frequency of the communication is relatively low and the computation to communication ratio is high. For the particular configuration that was used, the normal run (on good nodes with no failures) would last just around 100 seconds for SoE while it is around 4.5 hours for REMD. Both the applications were run 10 times in each scenario and the scenarios were constructed based on all the combinations of parameters listed below, with exceptions noted in the description of results. For REMD, all runs for a single scenario will typically complete in 50 to 100 hours. The number of runs was cut short in one case as the amount of time being consumed was excessive as is pointed out in the corresponding discussion.

The execution scenarios were combinations of the following applications and parameters:

1) REMD and SoE with No replicas, 2 replicas (i.e. 2 process instances) , and 3 replicas.
2) REMD and SoE with and without threshold based node selection.
3) REMD with a nominal checkpoint interval of 15 mins and 1 hour.

D. Experimental results

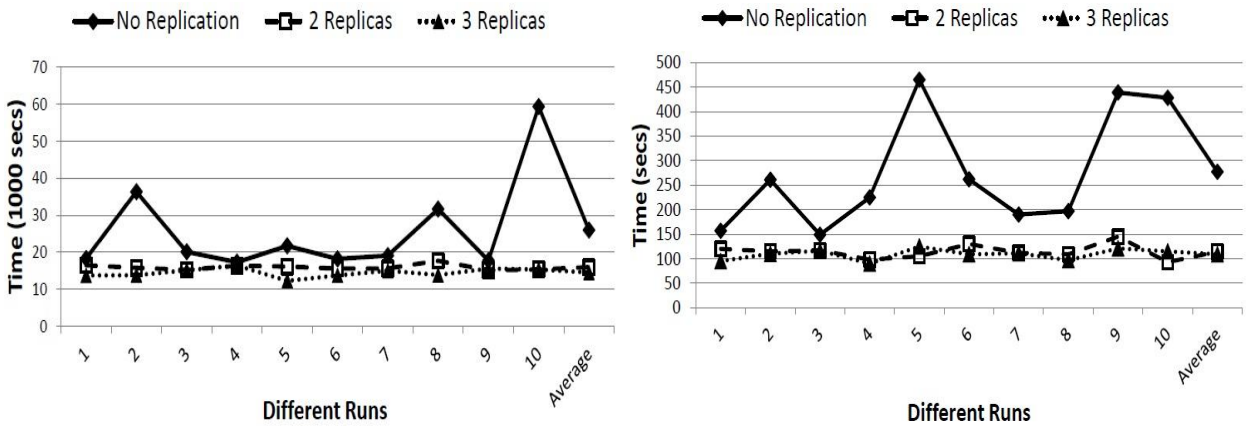
We present a suite of experimental results. Figure 9 shows the mean, median, and standard deviation for the set of 10 readings for each scenario for REMD, except for the case of 3 replicas. We now present graphs showing individual results from 10 execution runs with one scenario for one program. As expected for a volunteer environment, the results sometimes vary dramatically from run to run.

Impact of replication: Figure 5 presents experimental results for SoE and REMD when executed with no replication, 2 replicas per process and 3 replicas per process. No host selection was employed and a fixed checkpoint interval of 15 minutes was used for REMD (checkpointing does not apply for SoE). It is clear from the figure that employing replication has a dramatically positive impact on the execution time and the variability of the execution time. However, increasing the degree of replication from 2 to 3 does not have a significant positive or negative effect. The impact of replication is qualitatively similar for SoE and REMD although the relative difference is higher for the shorter running SoE.

Impact of host selection: Figure 6 presents experimental results for SoE and REMD when executed with and without replication, and with and without host selection. When replication is employed, the degree of replication is 2. A checkpoint interval of 15 minutes was used for REMD, with no checkpointing for SoE.

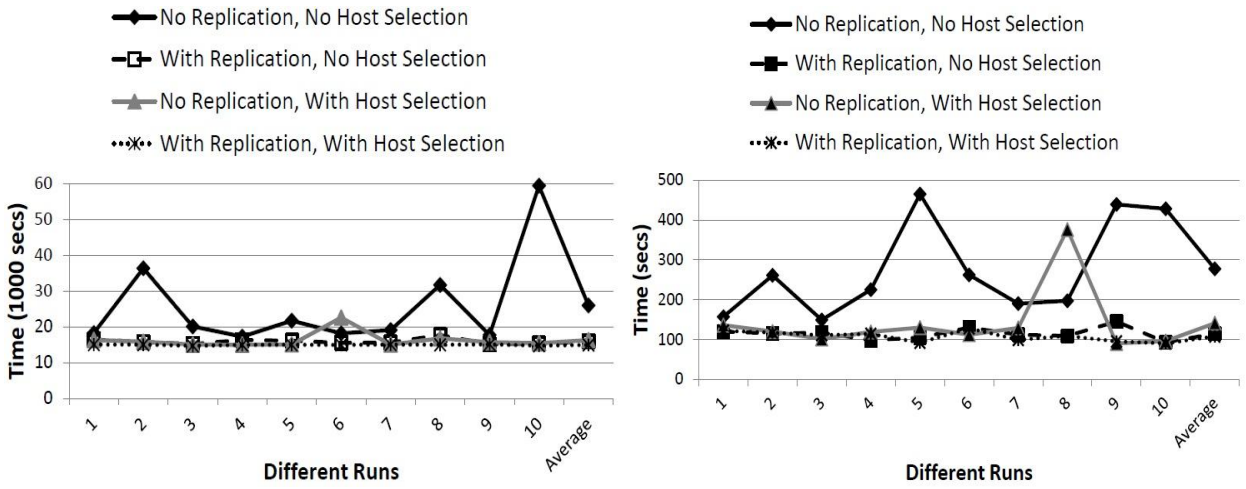
It can be seen from that figure that host selection improves the performance for both the programs. Node selection has a substantial positive effect on performance when no replication is used and reduces the range of execution times significantly. Now we focus on the following cases in relation to each other: node selection alone, replication alone, and node selection and replication together. It is difficult to visually tell the difference from the graph so we focus on the table in Figure 9. We note that the mean and the standard deviation are the highest for node selection alone (16.3,2.4), followed by replication alone(15.9, 0.7) followed by both (14.8, 0.11). The unit is 1000s of seconds. The overall conclusion is that both node selection and replication have a substantial positive effect individually and their cumulative effect is a small but clear improvement over using either one of them alone. Of the two,

[1]The term 'temperature replica' has an application specific meaning in the context of REMD, unrelated to Volpex process replicas.



(a) REMD (b) SoE

Fig. 5. REMD and SoE with different levels of replication. There is no host selection and the checkpoint interval for REMD is 15 mins



(a) REMD (b) SoE

Fig. 6. REMD and SoE with and without replication and host selection. The checkpoint interval for REMD is 15 mins

replication has a slightly stronger impact in terms of reducing the magnitude and variance of execution time.

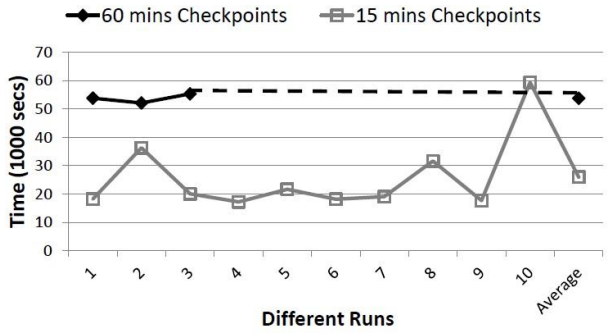


Fig. 7. REMD with different checkpoint intervals

Impact of Checkpoint interval: Experiments shown in Figure 7 were done for REMD without replication or host selection and the checkpoint intervals of 15 minutes and 1 hour. For the latter case, the execution times were consistently very high, more than 14 hours for each run when the nominal runtime was around 4.5 hours. The full set of experiments was abandoned after 3 runs. The reason is that a single host failure can lead to a delay around 1 hour and we certainly expect many failures in an execution. Clearly a well chosen checkpoint interval is very important in a volunteer environment, especially if good node selection and replication is not available.

Figure 8 examines the impact of the checkpoint interval value when there is replication, with two replicas per process. The cases with and without host selection are covered. All the lines in the graph seem close to each other, so it appears that in the presence of replication, making checkpointing less frequent has a very small effect. By looking closely at Figure 9 we observe that the mean execution time does not change in any meaningful way when the checkpoint interval increases from 15 mins to 60 mins. The standard deviation of execution

time increases very slightly when the checkpoint interval is increased from 15 mins to 60 mins; from .11 to .33 for the case with host selection and from 0.7 to 1.1 for the case with no host selection; the unit is 1000s of seconds. However, these are very small changes so the observation is that the execution time is virtually unaffected with increased checkpoint interval when replication is deployed. Another way to interpret the result is that replication allows us to have a longer checkpointing interval with no negative impact. For REMD, the size of the checkpoints is very small as only the execution parameters (like energy, temperature) need to be restored. An application with large footprint checkpoints could benefit significantly with a reduced frequency of checkpoints made possible by host selection and replication.

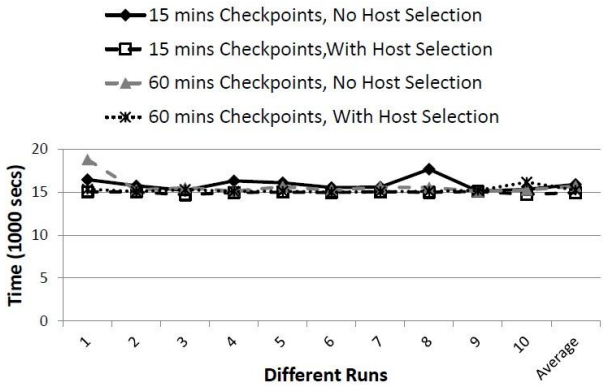


Fig. 8. REMD with checkpoints of 15 and 60 mins, with and without host selection. There is replication of degree 2 in all cases

VIII. Discussion

The experimental results clearly demonstrate that meaningful execution of long running parallel jobs in a volunteer environment is possible only with some combination of host selection, replication, and checkpointing. Application of multiple techniques yields more benefits than employing a single method, but the additional benefits are minimal in some cases. Checkpointing interval is not a critical issue when host selection and/or replication are also deployed. Host selection and replication are important even when checkpointing is used. As compared to host selection, replication seems to have a slightly higher beneficial effect.

The experiments represent over a month of execution on a real volunteer testbed. While the evaluation is still limited, the results provide significant insight into performance of other applications and on larger testbeds. We also note that different methods have different costs; replication doubles the resources that are used, node selection limits the number of nodes that are useful, and checkpointing can be expensive with the cost being strongly application specific.

Following are some of the directions of future work in this project:

- Further analysis of the impact of the techniques discussed. This will include experiments with a larger suite of applications, a larger set of parameters such as checkpoint intervals, and a larger testbed and application size.
- Evaluating the impact of more sophisticated scheduling and host selection schemes that identify and benefit from long range host availability patterns.
- Enhancing the execution framework to identify and correct Byzantine faults caused by hardware/software errors or by hackers.
- Full automation of checkpointing and replication, such that the checkpointing interval and the degree of redundancy are optimized dynamically during execution.
- Migration of a suite of parallel scientific codes to use volunteer computing with Volpex and BOINC.

IX. Conclusions

The central contribution of this paper is an execution environment that is able to effectively convert a volunteer PC grid into a virtual cluster for the execution of communicating parallel applications. Replication, checkpointing and host selection are combined and integrated in new ways to meet the daunting challenge of effective parallel execution on failure prone and heterogeneous nodes with dynamically changing execution behavior. Key innovations include fully autonomous replicas, unified replication and checkpointing, co-operative checkpointing across process replicas, hot spares for quick restarts, termination and restart of lagging processes, and high level user control over resource and performance tradeoffs. This work provides the first implementation and evaluation of the combination of these methods for volunteer computing with a real-life testbed. The system has been integrated with BOINC middleware to provide, to the best of our knowledge, the first comprehensive solution to the reliable execution of general parallel programs on volunteer nodes. The principles and methods developed in this work are applicable to other heterogeneous environments including computation clouds.

This research enables the use of volunteer computing to a significantly broader class of scientific codes than the state-of-the-art. The work also represents an actual execution platform that is free and freely available to scientists. We hope that this publication will serve as a medium for the scientists to contact us to deploy Volpex to achieve their scientific goals.

Acknowledgments

Partial support for this work was provided by the National Science Foundation's Computer Systems Research program under Award No. CNS-0834750 and MCB-0919974. 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. We would like to acknowledge the contributions of Keith Crabb at UH Research Computing Center and Tsung-I "Mark" Huang at Texas Learning and Computation Center towards providing the infrastructure for this project. Latoya Jackson performed the initial experiments that helped drive this research.

	No Host Selection No Replication Time (1000 secs)			No Host Selection With Replication Time (1000 secs)			With Host Selection No Replication Time (1000 secs)			With Host Selection With Replication Time (1000 secs)		
Checkpoint Interval	Mean	Median	Std.dev.	Mean	Median	Std.dev.	Mean	Median	Std.dev.	Mean	Median	Std.dev.
15 mins	25.9	19.5	13.4	15.9	15.6	0.7	16.3	15.7	2.4	14.8	15	0.11
60 mins	53.7	53.8	1.6	15.7	15.3	1.1	16.2	16.1	1.1	15.2	15.1	0.33

Fig. 9. Table for Mean, Median, and Standard Deviation for different REMD execution scenarios

REFERENCES

[1] http://www.almaden.ibm.com/cs/tspaces/.

[2] R. Anand, E. Gabriel, and J. Subhlok. Communication Target Selection for Replicated MPI Processes. In *R. Keller, E. Gabriel, M. Resch, J. Dongarra (Eds.) Recent Advances in the Message Passing Interface, LNCS 6305*, pages 198–207, Stuttgart, Germany, September 2010.

[3] R. Anand, T. LeBlanc, E. Gabriel, and J. Subhlok. A Robust and Efficient Message Passing Library for Volunteer Computing Environments. *Journal of Grid Computing*, 9(3):325–344, 2011.

[4] D. Anderson, C. Christensen, and B. Allen. Designing a Runtime System for Volunteer Computing. In *Proceedings of the Supercomputing (SC)*, page 33, 2006.

[5] D. P. Anderson. BOINC: A system for public-resource computing and storage. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[6] A. Andrzejak, D. Kondo, and D. Anderson. Exploiting non-dedicated resources for cloud computing. In *Proceedings of the Network Operations and Management Symposium (NOMS)*, pages 341 – 348, 2010.

[7] R. Batchu, J. P. Neelamegam, Z. Cui, M. Beddhu, A. Skjellum, and Y. Dandass. MPI/FT: Architecture and taxonomies for fault-tolerant, message-passing middleware for performance-portable parallel computing. In *In Proceedings of the 1st IEEE International Symposium of Cluster Computing and the Grid*, pages 26–33, 2001.

[8] 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 *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 25, Washington, DC, USA, 2003. IEEE Computer Society.

[9] N. Carriero and D. Gelernter. The S/Net's Linda kernel. *ACM Trans. Comput. Syst.*, 4(2):110–129, 1986.

[10] D. Case, D. Pearlman, J. W. Caldwell, T. Cheatham, W. Ross, C. Simmerling, T. Darden, K. Merz, R. Stanton, and A. Cheng. *Amber 6 Manual*. 1999.

[11] C. Docan, M. Parashar, and S. Klasky. DataSpaces: An Introduction and Coordination Framework for Coupled Simulation Workflows. In *Proceedings of the High Performance Distributed Computing (HPDC)*, pages 25–36, 2010.

[12] P. Domingues, J. Silva, and L. Silva. Sharing checkpoints to improve turnaround time in desktop grid computing. In *Proceedings of the Advanced Information Networking and Applications (AINA)*, page 6, 2006.

[13] J. Duell, P. Hargrove, and E. Roman. The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart. In *Berkeley Lab Technical Report (publication LBNL-54941)*, 2002.

[14] T. Estrada, M. Taufer, and D. P. Anderson. Performance prediction and analysis of BOINC projects: An empirical study with EmBOINC. *Journal of Grid Computing*, 7(4):537–554, 2009.

[15] K. Ferreira, R. Riesen, R. Oldfield, J. Stearly, J. Laros, K. Redretti, T. Kordenbrock, and R. Brightwell. Increasing fault resiliency in a message-passing environment. Technical report, Sandia National Laboratories, 2009.

[16] S. Genaud and C. Rattanapoka. 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*, pages 1–8, 2008.

[17] J. Hursey, T. I. Mattox, and A. Lumsdaine. Interconnect agnostic checkpoint/restart in Open MPI. In *HPDC '09: Proceedings of the 18th ACM international symposium on High Performance Distributed Computing*, pages 49–58, New York, NY, USA, 2009. ACM.

[18] B. Javadi, D. Kondo, J. Vincent, and D. Anderson. Discovering Statistical Models of Availability in Large Distributed Systems: An Empirical Study of SETI@home. In *Proceedings of the Parallel and Distributed Systems (TPDS)*, page 1, 2011.

[19] N. Kanna, J. Subhlok, E. Gabriel, and D. Anderson. A communication framework for fault-tolerant parallel execution. In *Proc. The 22nd International Workshop on Languages and Compilers for Parallel Computing*, Newark, DE, 2009.

[20] D. Kondo, A. Andrzejak, and D. P. Anderson. On correlated availability in internet-distributed systems. In *9th IEEE/ACM International Conference on Grid Computing (Grid 2008)*, Tsukuba, Japan, Sep 2008.

[21] D. Kondo, M. Taufer, C. Brooks, H. Casanova, and A. Chien. Characterizing and evaluating desktop grids: an empirical study. *Proceedings. 18th International Parallel and Distributed Processing Symposium*, page 26, April 2004.

[22] H. Lin, X. Ma, J. Archuleta, W.-C. Feng, M. Gardner, and Z. Zhang. DataSpaces: An Introduction and Coordination Framework for Coupled Simulation Workflows. In *Proceedings of the High Performance Distributed Computing (HPDC)*, pages 25–36, 2010.

[23] M. S. Noble and S. Zlateva. Scientific computation with javaspaces. In *HPCN Europe 2001: Proceedings of the 9th International Conference on High-Performance Computing and Networking*, pages 657–666, London, UK, 2001. Springer-Verlag.

[24] E. Pedamallu, H. Nguyen, N. Kanna, Q. Wang, J. Subhlok, E. Gabriel, M. Cheung, and D. Anderson. A robust communication framework for parallel execution on volunteer PC grids. In *CCGrid 2011: The 11th IEEE/ACM International Symposium on Clusters, Cloud and Grid Computing*, Newport Beach, CA, May 2011.

[25] X. Ren and R. Eigenmann. iShare - Open internet sharing built on peer-to-peer and web. In *European Grid Conference*, Amsterdam, Netherlands, Feb 2005.

[26] B. Rood and M. J. Lewis. Availability Prediction Based Replication Strategies for Grid Environments. In *Proceedings of the Cluster, Cloud and Grid Computing (CCGrid)*, pages 25 – 33, 2010.

[27] S. Sankaran, J. M. Squyres, B. Barrett, A. Lumsdaine, J. Duell, P. Hargrove, and E. Roman. The LAM/MPI checkpoint/restart framework: System-initiated checkpointing. *International Journal of High Performance Computing Applications*, 19(4):479–493, Winter 2005.

[28] Y. Sugita and Y. Okamoto. Replica-exchange molecular dynamics method for protein folding. *Chemical Physics Letters*, 314:141–151, 1999.

[29] M. Taufer, A. Kerstens, T. Estrada, D. Flores, and P. Teller. SimBA: a discrete event simulator for performance prediction of volunteer computing projects. In *International Workshop on Principles of Advanced and Distributed Simulation 2007*, march 2007.

[30] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[31] Q. Wang, K.-C. Liang, A. Czader, M. N. Waxham, and M. S. Cheung. The effect of macromolecular crowding ionic strength and calcium binding on calmodulin dynamics. *PLoS Comput Biol*, 7(7), 2011.