



VoIPEX

Failsafe MPI Program Execution

Troy P. LeBlanc

May 17, 2007



Agenda

- Parallel Execution on Volatile Nodes
- Overview and Approach
- Literature Survey
- Models
- Research Challenges
- VolPEX Implementation
- Preliminary Experiments
- Development
- Conclusion

Motivation – Solving the large, truly parallel scientific problems using essentially free computing resources.



Parallel Execution on Volatile Nodes

■ Parallel Computing

- The simultaneous execution of a task on multiple processors in order to obtain results faster
- E.g., Weather prediction, Oil Exploration, Protein Analysis
 - Clusters are highly reliable
 - Problem - Very expensive computing resources

■ ...on Volatile Nodes

- Volunteer computing is an arrangement in which people (volunteers) provide computing resources to projects, which use the resources to do distributed computing
 - Essentially free computing resources
 - Problem – Volunteer systems run at the speed of the slowest node.
- Volatile Node - A volunteer computer can/will become unavailable at ANY time!



Overview

- We propose to develop a framework that allows reliable execution of communicating parallel programs on volatile (volunteer) nodes.
- Execute ‘unaltered’ MPI programs on volunteer nodes in a failsafe manner

Approach

- Redundant copies of each volunteer “process”
- Avoid redundant data transfer
- Progress at the speed of the fastest nodes
- Minimize/Avoid blocking during Message Exchange



Literature Survey

- Public Resource or Volunteer Computing
 - Master/Slave paradigm – bag of (relatively) independent tasks
 - BOINC - SETI@home, Predictor@home; IBM World Community Grid
 - Use public resources, but have no slave interaction

- Fault-Tolerant MPI Installations
 - MPI/FT – built to handle transient node failure
 - Adaptive MPI – geographically relocating MPI processes
 - PC³ – thin layer that handles physical node mapping
 - EasyGrid - dynamic scheduling middleware across clusters
 - These are attempts to allow use of MPI in “larger” clusters
 - Flavors of MPI
 - C, C++, FORTRAN, Python, JAVA, C#(.NET), Perl



Literature Survey (cont.)

- Peer-to-Peer Systems (cycle-sharing frameworks)
 - Well known P2P today are mostly data-sharing frameworks
 - BambooTrust – providing volunteer node security
 - WaveGrid – timezone aware network overlay
 - Phoenix – dynamic routing table, transparent process migration
 - Each uses P2P in Global Computing, current emphasis on resource scheduling, none provide access to MPI libraries



Models

■ 3 framework models

1. Reliable Intermediate (MS Thesis-Kutiev)

- Intermediate node having message buffers

2. PUSH Model

- Send is immediate, nodes have Recv buffers

3. GET Model

- Receive is immediate, nodes have Send buffers

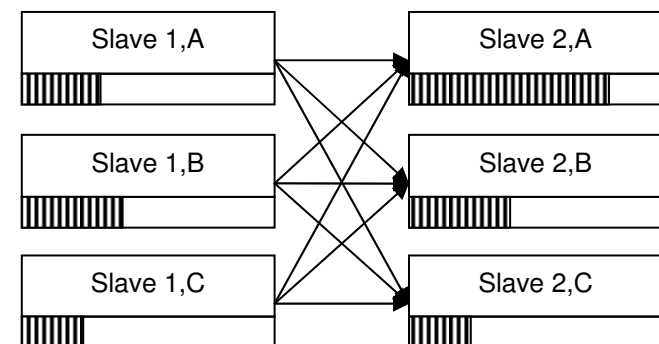
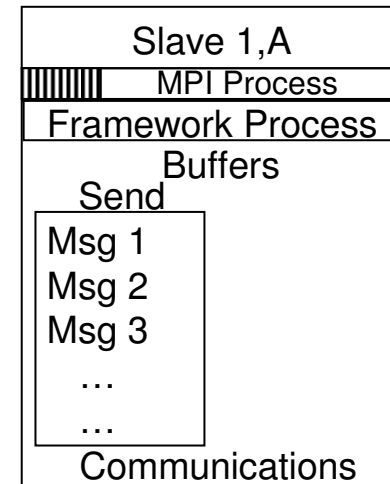
- Criteria to be studied

- Buffer Management, # of communications, overall application progress

Models (cont.)

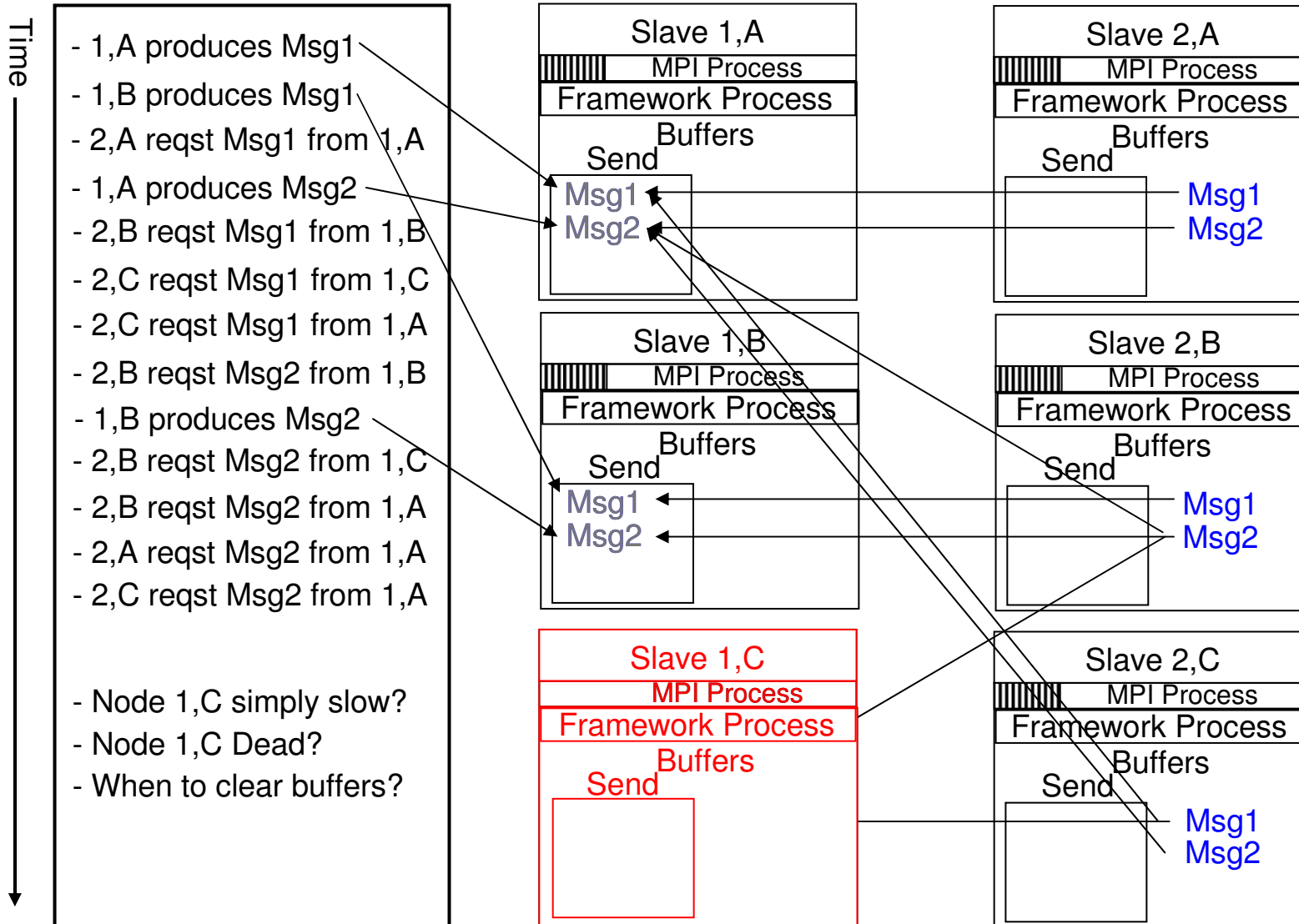
■ GET Model

- A1 produces some data
 - MPI_Send intercepted locally to place data in send buffer
- B1 GETS data from A1
 - MPI_Recv intercepted and occurs immediately if data is in remote buffer
- a. Buffer Management
 - Decision required for when to delete data in Send Buffer
- b. # of communications (Min 1/Max 3 RTT)
 - B1 GET from A1 on MPI_Recv()
 - If(SlaveHealth(A1))
 - MPI_Send(Msg1 from A1)
 - else if(SlaveHealth(A2))
 - MPI_Send(Msg1 from A2)
 - else if(SlaveHealth(A3))
 - MPI_Send(Msg1 from A3)
- c. Impact on fastest nodes
 - Should B2 check with A1 first?
- d. Impact of transient slowness
 - Complicates decision when to delete data in Send Buffer (transient slowness)



GET Model Example

Timeline





Research Challenges

- Scalability – performance as framework size increases
 - How far can we push framework size?
- Overhead
 - Computation to communication ratio
- Resource Management
 - Tracking and prioritizing most viable volunteers
 - Tracking and maximizing most idle time
 - Adhering to user preferences for availability
- Connectivity, Bandwidth, and Latency
 - characterize framework's use of network
- Replica Management
 - Number required, Selection, Mapping, Termination and Regeneration



VoIPEX Implementation

- High-level

1. Node Management Service

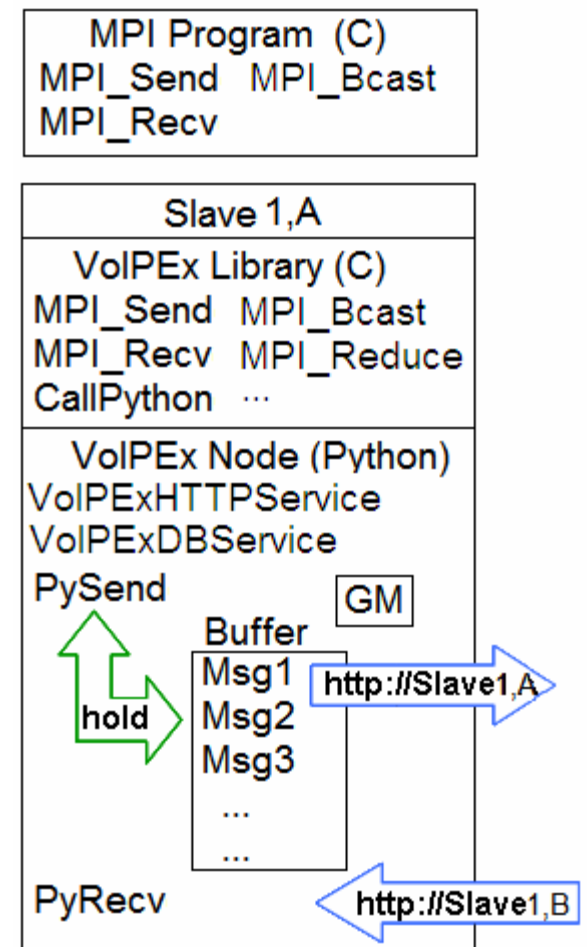
- Maintain a global mapping of the volunteer slave nodes
- Slave Health statuses
- Create Node Mapping to manage redundancy
- Buffer Mngmt

2. Messaging Service

- Network communications protocol between nodes
- Uses GET Model
- Each node runs web services
- VoIPEX MPI Library

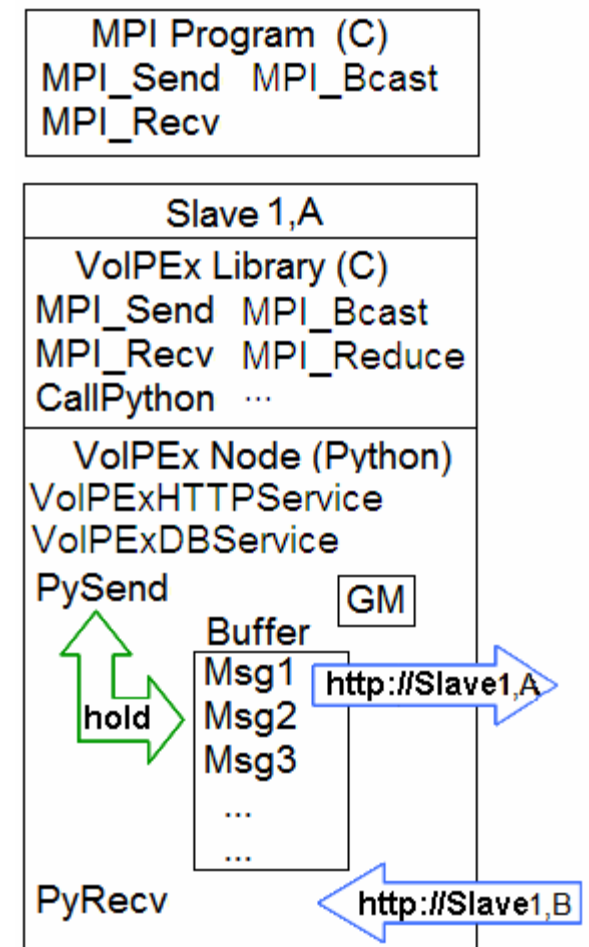
VoIPEX Implementation (cont.)

- Node Management Service (Python)
 - Master is always ON – reliable
 - On start-up, slave nodes register
 - Existing Global Map entry, or
 - New Global Map entry
 - Slave nodes HealthCheck (check-in) with the Master Node
 - Slaves check in every 30 seconds
 - Master gives each slave a status based on last check-in time
 - < 30 seconds **active**
 - 30 to 300 seconds **suspect**
 - > 300 seconds **inactive**
 - Other statuses
 - avail, start, barrier, done
 - Python DB service, in-memory
 - Global Map, Send Buffer, Event Log



VoIPEX Implementation (cont.)

- Messaging Service
 - VoIPEX Library (C)
 - MPI_Init, MPI_Finalize
 - MPI_Comm_rank, MPI_Comm_size
 - MPI_Send, MPI_Recv, MPI_Isend
 - More in upcoming slide
 - Multi-threaded web service
 - Allows only very specific url requests
 - All communication transfer string data via HTTP GET with CGI scripting
 - Datatype (Int, Float, Double and Integer, Real, Double_Precision) arrays are converted to strings for transfer, and re-converted at destination to appropriate datatype array



Development (cont.)

- NAS Benchmarks use the following MPI functions
 - Ranked by percentage use across the benchmarks

MPI Function		Frequency (%)	MPI Function		Frequency (%)
MPI_Irecv	(C,F)	14.4%	MPI_Finalize	(C,F)	3.0%
MPI_Send	(C,F)	10.6%	MPI_Comm_rank	(C,F)	2.5%
MPI_Isend	(C,F)	10.2%	MPI_Init	(C,F)	2.5%
MPI_Bcast	(C,F)	9.7%	MPI_Reduce	(C,F)	2.5%
MPI_Wait	(C,F)	9.7%	MPI_Alltoall	(C,F)	1.7%
MPI_Allreduce	(C,F)	7.2%	MPI_Comm_dup	(C,F)	1.7%
MPI_Barrier	(C,F)	7.2%	MPI_Comm_split		1.7%
MPI_Abort	(C,F)	4.7%	MPI_Recv	(C,F)	1.7%
MPI_Comm_size	(C,F)	4.2%	MPI_Wtime	(C,F)	0.8%
MPI_Waitall	(C,F)	3.4%	MPI_Alltoallv	(C,F)	0.4%

VoIPEx Implementation (cont.)

VoIPEx - Windows Internet Explorer

http://129.7.248.116/~tpleblanc/cgi-bin/cgiMASTER.py

File Edit View Favorites Tools Help Links CNN KHOU Google JSC Webmail VoIPEx JSC-VPN MPI Functions MPI.DEINO NASALL

bash command line argument... VoIPEx

VoIPEx - Failsafe MPI Program Execution

UNIVERSITY of HOUSTON

Home Master Node Downloads Current Research Future Development Publications Statistics Partnerships Related Work Biographies Documentation Comments

Upload Source: Browse...
 Slaves Req'd: Set Mapping Upload C
 Redundancy (1-10): Reset Mapping Upload F
 Start! Delete Output

Master (129.7.248.116) Refresh Views

Global Map

Public IP	Checked in @	Status	Communicators				
129.7.248.116	136/10:40:32	avail	0,M	none	none	none	none
70.240.226.241	136/10:40:24	avail	1,A	none	none	none	none

Send Buffer

Data	Count	DataType	Destination	Tag	Comm
0	0	0	0	0	0
-801.878454,	1	15	0	61000	1
-3247.834652,	1	15	0	60000	1
-5083.561080,	1	15	0	61001	1
-6958.407078,	1	15	0	60001	1
3069581.000000,2931822.000000,5	10	15	0	61002	1
6140517.000000,8798778.000000,1	10	15	0	60002	1
23.000000,	1	15	0	61003	1
23.000000,	1	15	0	60003	1

Event Log

Time	Event
136/10:39:52.278	MPI Process ID is 28743
136/10:39:52.416	Initializing MPI Program
136/10:39:52.416	Initializing MPI Program

Output (Master Node)

© 2007

Done Internet 100%



Preliminary Experiments

- Compare runs of MPI programs on VolPEX and MPICH-1.2.6 for 1st prototype for Preliminary Defense
 - t0.c – MPI_Init, MPI_Finalize
 - t1.c – MPI_Comm_rank, MPI_Comm_size
 - t2.c – MPI_Bcast
- VolPEX records all events to millisecond
- MPI Profiling library written to catch Init and Finalize to millisecond for MPICH runs
- For 2nd prototype, continue adding MPI functions to conduct similar tests using NAS and other benchmarks
 - e.g., compiled and executed EP this week
 - Expect to run all NPB by end of summer



Development

- VolPEX Prototype v.1 Fall '06
 - MPI Functions (Init, Finalize, Rank, Size, Send, Recv, Bcast, Reduce)
 - Handles arrays of integers, floats, doubles
 - Utilizes node redundancy
 - Test simple unaltered MPI C executables

- VolPEX Prototype v.2 Spring '07
 - Work with NATs/Firewalls (succeeded with HTTP over SSH Tunnel)
 - Heterogeneous nodes (have Linux and Windows Python nodes running)
 - Deployment methodologies
 - Add more MPI Functions and datatypes
 - Test unaltered MPI Fortran executables (compiled with VolPEX MPI Library)
 - Test NAS and other common benchmarks ----- Current
 - Realistic application (UH Air Quality or Molecular Biology groups)
 - Checking correctness of results (checksums)
 - Security against web service attacks
 - PUSH Model
 - Resource Mngmt (node selection by idle time, user prefs, etc.)
 - BOINC Integration
 - UDP Implementation (vs. TCP or HTTP over TCP)
 - Checkpointing

Development (cont.)

■ Work with NATs/Firewalls

- Python functions return both the private and public IP address of a computer
 - Publicly broadcast IP from the router
- For slave-to-slave communication want to allow direct HTTP requests of each other.
- Possible solutions
 - Reliable Intermediate node
 - HTTP over SSH Tunnel
 - GCB: Generic Connection Brokering
- HTTP over SSH Tunnel
 - OpenSSH daemon on Windows hosts (Linux hosts already installed)
 - RSA Public Key Authentication (exchange public keys)
 - Define Local Port Forwarding in a config file on the receiving node
- Dealing with DHCP
 - Keep the publicly advertised IP up-to-date in the SSH config file and SSH known_hosts file

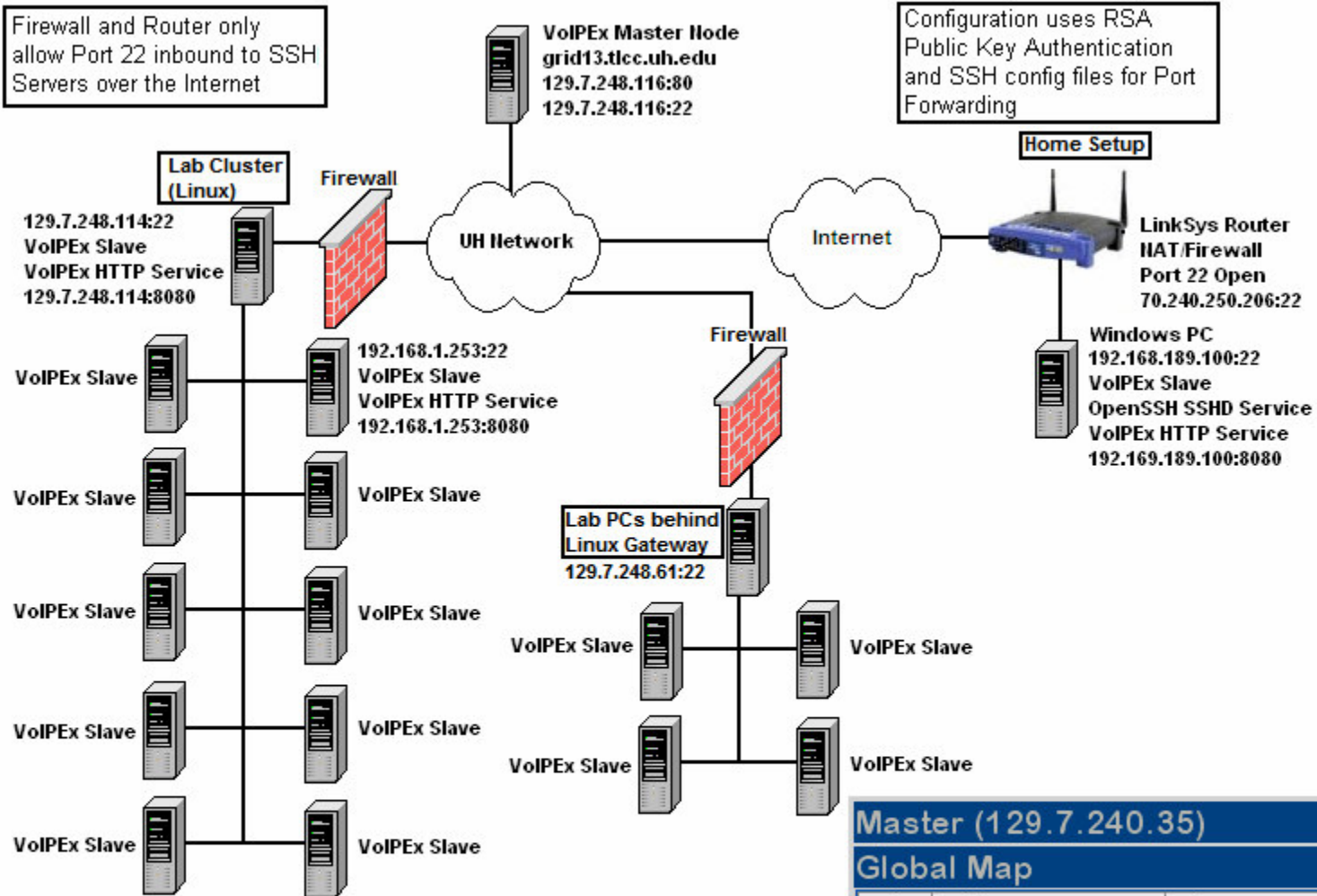
```
Host home
HostName 70.240.243.219
  User VolPEX
  LocalForward 19000 192.168.189.100:8080
  LocalForward 19001 192.168.189.100:22

Host frontend
HostName 129.7.248.114
  User tpleblanc
  LocalForward 19998 129.7.248.114:8080
  LocalForward 19999 129.7.248.114:22
```

NATs/Firewalls

Firewall and Router only allow Port 22 inbound to SSH Servers over the Internet

Configuration uses RSA Public Key Authentication and SSH config files for Port Forwarding



Master (129.7.240.35)					
Global Map					
Index	Public IP	Private IP	Map	Status	Group
0	129.7.240.35	129.7.241.1	0,M	active	none
1	129.7.248.114	129.7.248.114	0,S	active	none
2	70.240.250.206	192.168.189.100	0,S	active	none



Development (cont.)

- Deployment methodologies
 - As the Framework gets larger, will need to allow for automated updates
 - Updates to nodes on NFS was easy; however, individual restarts
 - Updates to independent nodes (Windows) are individual as well as restarts
 - We want to leave all nodes (esp. the Master) running
 - ~8MB RAM and Minimal processor time for periodic check-in
 - Python custom solution
 - Nodes Check-in for latest updates
 - Reload() function allows VolPEX to utilize updated functions
 - We are investigating BOINC Services



Development (cont.)

- Heterogeneous nodes (adding Windows nodes)
 - Linux Apache Web Server will host the reliable Master node
 - Deploy VoIPEx Slave node on Windows and Linux PCs
 - Python Interpreter freely available for many platforms
 - Require compiled MPI executables for target hosts

- Handling C and FORTRAN MPI programs
 - Linux has gcc and gfortran available
 - MingW compiler on Windows has gcc and g95 available
 - Cross-compilers may be an option
 - MPI Specifications use the C library to handle both C and FORTRAN MPI programs
 - #pragma weak function names (updates symbol table for linker)
 - Variable length argument lists (...) handle choosing call-by-reference or call-by-value and any additional arguments
 - Arrays must be correctly aligned



Conclusion

- VolPEX prototype is nearly complete
 - Python framework that runs on Linux and Windows platforms
 - Handles both C and FORTRAN MPI programs
 - Runs in background (don't yet have data on resource usage!)
- New approach to parallel computing on volunteer computers
 - Mapping redundant slave nodes provides failsafe MPI communication and execution
 - Allows use of ordinary volunteer computers for MPI programs
- Application Domain will likely be loosely coupled parallel computations
- Example MPI programs or other benchmarks would be welcome!