

# Rapid Development of Parallel Systems and Applications in River

April 17, 2007

Greg Benson

Alex Fedosov, Brian Hardie, Tony Ngo, Yiting Wu, Jennifer Reyes, Joseph Gutierrez



BayBUG 2007

# What is River?

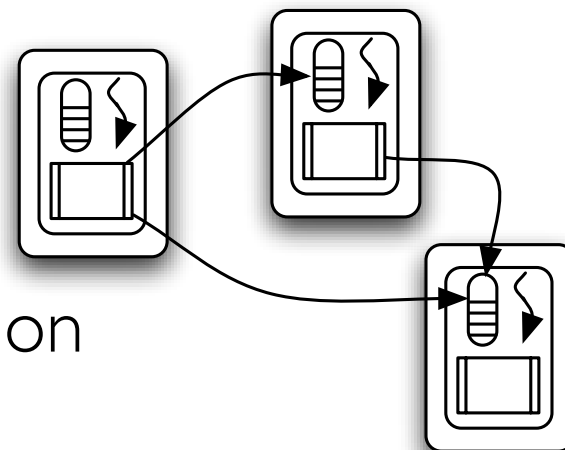
---

- ▶ **R**eliable **V**irtual **R**esources
- ▶ A Python framework for parallel and distributed programming
  - ▶ Used to write parallel Python programs
  - ▶ Used to prototype parallel programming systems

# River Overview

---

- ▶ River Core
  - ▶ Discovery
  - ▶ Process naming and creation
  - ▶ Message passing
  - ▶ State management
- ▶ River Extensions
  - ▶ MPI, Trickle, MapReduce



# River Benefits

---

- ▶ Small, easy to use core interface
- ▶ Written entirely in Python
- ▶ Dynamic typing for rapid prototyping
- ▶ Python goodies
  - ▶ Heterogeneous (Use Python as a VM)
  - ▶ State capture at language VM level
  - ▶ Integrated checkpointing and migration

# Motivation

---

- ▶ Parallel programming is still hard
- ▶ The future: more cores, larger clusters
- ▶ Apps will have to utilize multiple processors
- ▶ Apps will have to tolerate failures
- ▶ The quest:
  - ▶ Find the next set of programming models
  - ▶ Incrementally improve current models

# Parallel Run-time Systems

---

- ▶ Developing a parallel language or interface:
  - ▶ Specify constructs and semantics
  - ▶ Implement compiler and run-time system
- ▶ My experience
  - ▶ SR (on UNIX, Amoeba, raw HW)
  - ▶ USFMPI (multi-threaded, TCP/Myrinet)

# Current Practice

---

- ▶ Design/development cycle (long)
  - ▶ Specify (perhaps by committee or group)
  - ▶ Prototype (Use C/C++/Java)
  - ▶ Use prototype to provide feedback
- ▶ Examples
  - ▶ MPI, X10, Fortress

# Constraining Designs

- ▶ Early implementation decisions become hard to undo
- ▶ Examples:
  - ▶ USFMPI internal formats
  - ▶ USFMPI dynamic thread model support
  - ▶ MPICH the reference implementation
  - ▶ Fault tolerance

```
struct MPI_Request_Int {
    int active; /* 0 if not active, 1 if active, 2 if persistent act */
    int done; /* Operation is complete */
    int type; /* 0 recv, 1 send, 2 probe*/
    int blocking; /* 0 non-blocking, 1 blocking */
    int posted; /* 0 post not sent, 1 post sent */
    void* buf; /* The message buffer */
    int size; /* The size of the buffer */
    int tag; /* The tag of the message */
    int peer; /* The other peer of communication */
    int comm; /* The communicator of the message */
    int multiple; /* 1 if multiple chunks, 0 if one chunk */
    int nchunk; /* Number of chunks sent or received */
    USFMPI_DD_type *stype; /* Pointer to derived datatype */
    int dcount; /* Counter used for send/receive */
    struct _USF_request_node* node; /* Back pointer for easy removal */
    int end_of_transmission ; /* to synchronize command and main thread */
}
```

# River Goals

---

- ▶ Extend Python's rapid development capabilities to parallel systems
- ▶ Facilitate short design/implementation cycles
  - ▶ Open up design space
- ▶ Enable prototypes to run on real HW
- ▶ Demonstrate scalability/feasibility

# Remainder of Talk

---

- ▶ A little Python
- ▶ The River Core
- ▶ Trickle (simple task farming language)
- ▶ River MPI (rMPI)
- ▶ Related and Future Work

# Python Features

- ▶ Built-in data types
- ▶ Concise syntax
- ▶ Dynamic typing
- ▶ Flexible objects
- ▶ Introspection
- ▶ Generators, list comprehensions

```
list = [3, 'foo', 2.4]
dict = { 'name': 'Alex', 'id': 4 }
```

```
def fact(n):
    if n == 1: return 1
    else: return n * fact(n-1)
```

```
class morph(object):
    def foo(self, x, y):
        return x + y
    def __getattr__(self, name):
        return self.foo
```

```
src = inspect.getsource(obj)
```

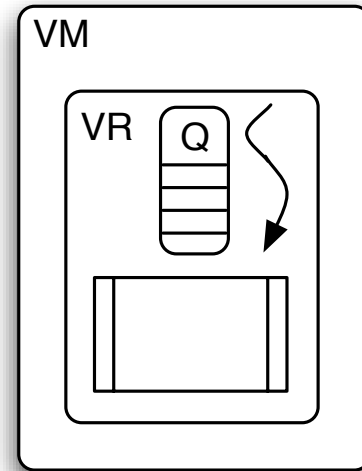
```
def factgen(n):
    for i in range(n):
        yield(fact(i))

print [x for x in factgen(10)]
[1, 1, 2, 6, 24]
```

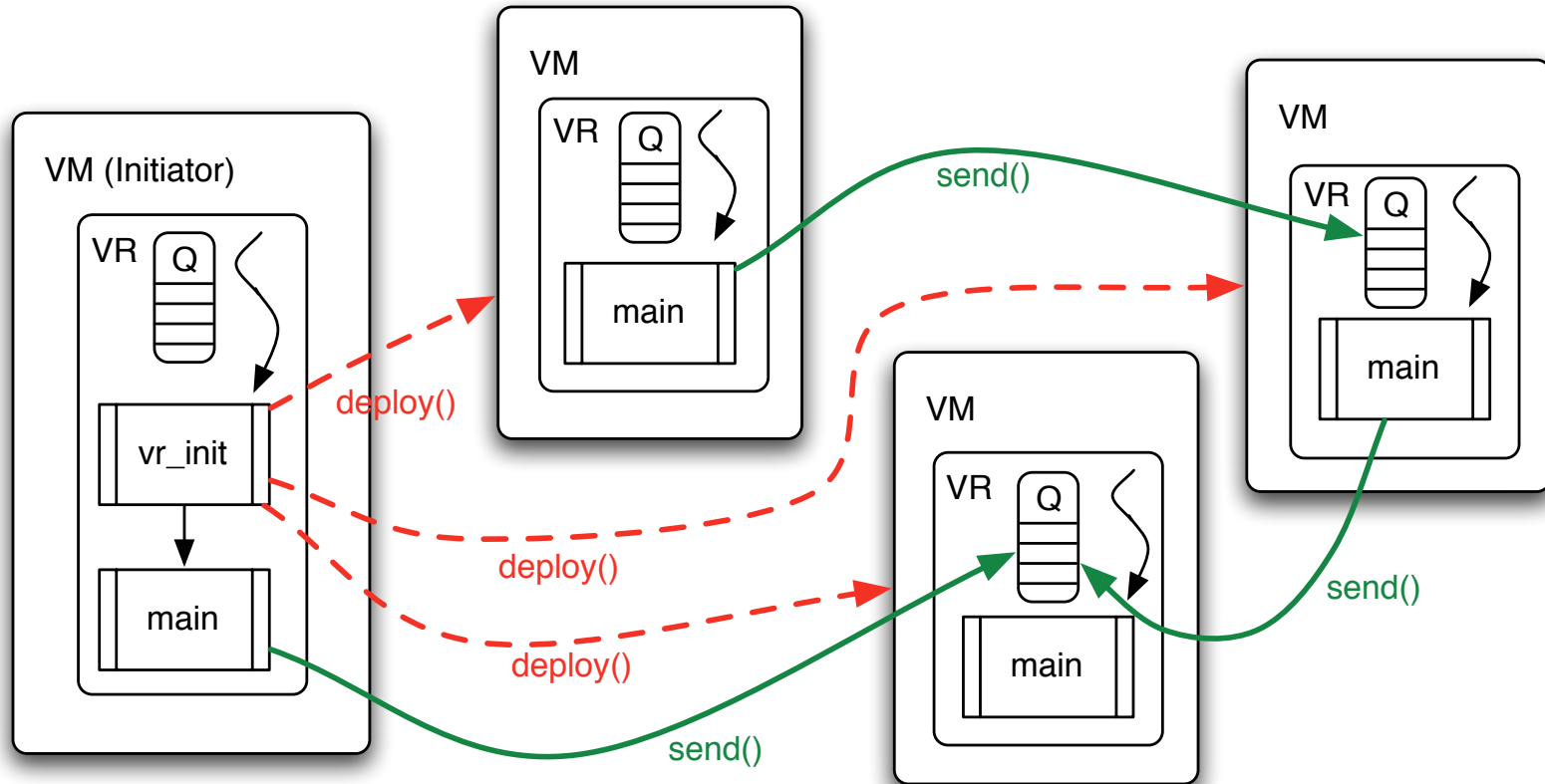
# River Concepts

---

- ▶ Virtual machines (VMs)
- ▶ Python + River Core
- ▶ Virtual resources (VRs)
  - ▶ Code, data, thread, and message queue
  - ▶ Named with UUIDs
- ▶ Discover/allocate/deploy



# Executing VRs



# Super Flexible Messaging

## ▶ Sending

```
send(dest=VRID, text='hello')
send(dest=VRID, tag='inputlist', items = [1, 2, 3, 4])

stk = Stack();stk.push(1); stk.push(2)
send(dest=VRID, data=stk)
```

## ▶ Receiving (selective)

```
m = recv()                # Any message
m = recv(tag='inputlist') # Specific attr and value
print m.items

m = recv(tag='inputlist', items=(lambda x:len(x) > 1))
m = recv(src=VRID, data=ANY)
print m.data.pop()
```

# Simple River Program

```
from socket import gethostname
from river.core.vr import VirtualResource

class simple (VirtualResource):
    def vr_init(self):
        discovered = self.discover()
        allocated = self.allocate(discovered)
        deployed = self.deploy(allocated, module=self.__module__)
        self.vrlist = [vm['uuid'] for vm in deployed]
        return True

    def main(self):
        if self.parent is None:
            for vr in self.vrlist:
                m = self.recv(src=vr)
                print m.myname
        else:
            self.send(dest=self.parent, myname=gethostname())
```

# Remote Invocation

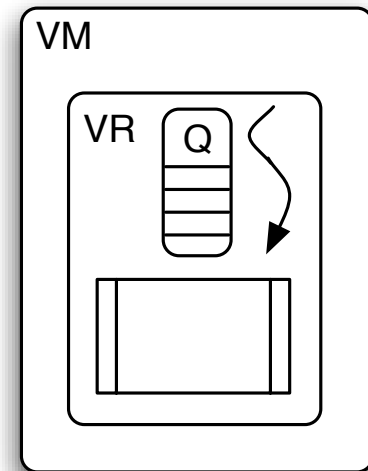
---

- ▶ Remote Access and Invocation (RAI)
  - ▶ RPC, RMI, and remote data access
  - ▶ Create and access functions, objects, data on remote VRs
  - ▶ Built on top of the River core
- ▶ Unrestricted mode

```
r = RemoteVR(server, self)
print r.add(1,2,3)
```

# State Management

- ▶ Designed from the beginning
- ▶ Based on Stackless Python
- ▶ Encapsulate local state in VR
- ▶ Only hooks to outside are UUIDs (soft)
- ▶ Per-VR queues hold in-transit messages
- ▶ Transparent migration and checkpointing



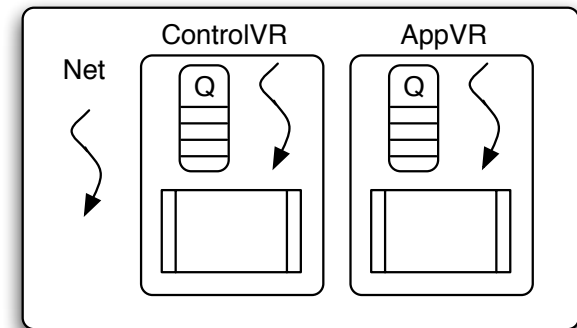
# Coordinated Checkpointing

---

- ▶ Algorithm
  - ▶ Freeze all remote VRs (preemptively)
  - ▶ Allow on the wire messages to settle
  - ▶ Write frozen state (VR + queue)
  - ▶ Unfreeze all remote VRs
- ▶ Mechanism is extensible
  - ▶ State exclusion, diskless, app assisted, etc.

# River Implementation

- ▶ One net thread, one Control VR
- ▶ Control handles VR creation, state
- ▶ TCP-based, connection caching (scalable)
- ▶ Broadcast-based discovery
- ▶ Super Flexible Messaging
  - ▶ Queue matching
  - ▶ Serialization (Pickle)



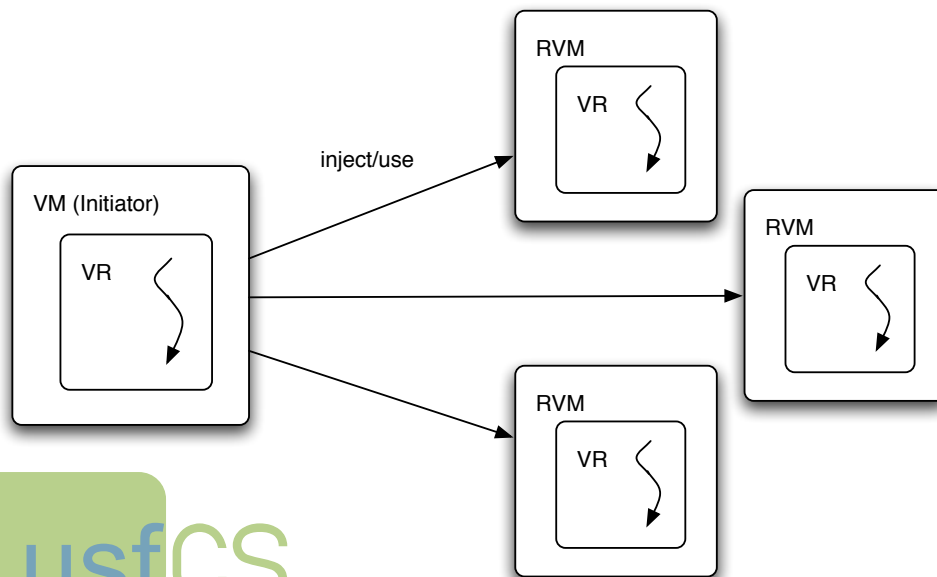
# River Complexity

---

<b>Component</b>	<b>LOC</b>
River Core	2,813
RAI (remote invocation)	514
Trickle (task farming)	373
rMPI	660
MapReduce	511

# Trickle

- ▶ Simple task farming language
- ▶ Put code/data on remote VMs
- ▶ Execute sequentially or in parallel



```
def foo(x):  
    return x + 10  
  
vmlist = connect()  
inject(vmlist, foo)  
results = [vm.foo(10) for vm in vmlist]  
print results  
  
$ trickle exsimple.py  
[trickle: discovered 4 VMs]  
[20, 20, 20, 20]
```

# Parallel Invocation

---

## ▶ Fork/join paradigm

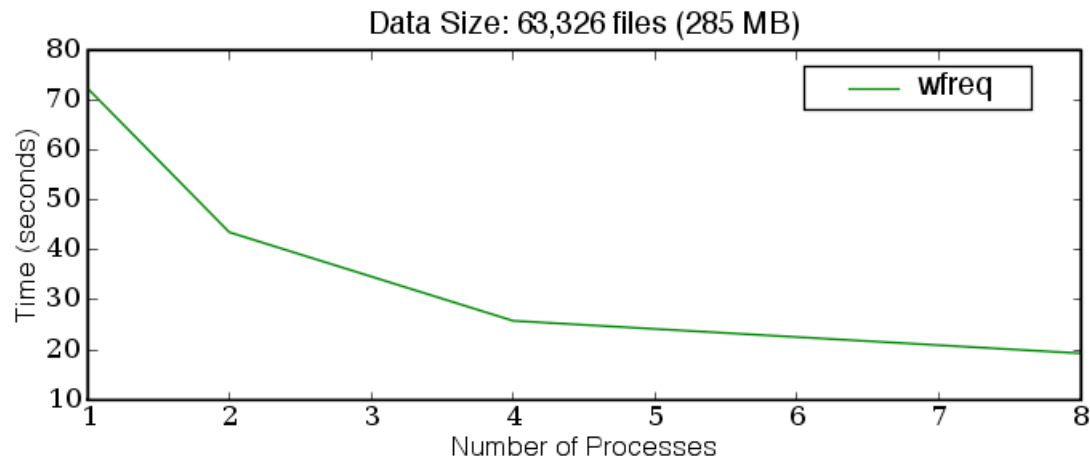
```
def foo(x):  
    return x + 10  
  
vmlist = connect()  
inject(vmlist, foo)  
hlist = fork(vmlist, foo, range(len(vmlist)))  
  
print join(hlist)
```

## ▶ Dynamic scheduling

```
def foo(x):  
    return sum(x)  
  
vmlist = connect()  
inject(vmlist, foo)  
results = forkwork(vmlist, foo, range(100), chunksize=10)  
  
print sum(results)
```

# Word Frequency

```
def wordcount(files):  
    # count words in given files (13 lines)  
def mergecounts(dlist):  
    # merge resulting count dictionaries (8 lines)  
  
# Command line processing (9 lines)  
  
vmlist = connect(n)  
inject(vmlist, wordcount)  
rlist = forkwork(vmlist, wordcount, files, chunksize=cs)  
final = mergecounts(rlist)
```



Penguin Cluster

AMD Opterons  
(Dual dual-core) 2.0GHz  
4 GB RAM, GigE

One VM per node

# River MPI (rMPI)

---

- ▶ Partial implementation of MPI 1.2 in River
- ▶ Most p2p and collectives
- ▶ Easy to read and understand
- ▶ Experiment with different algorithms
- ▶ Currently 660 lines of code

# rMPI Hello World

```
from mpi import *

class Hello( mpi ):
    def main( self ):

        self.MPI_Init()
        rank = mpi_Rank()
        np = mpi_Size()
        self.MPI_Comm_rank( MPI_COMM_WORLD, rank )
        self.MPI_Comm_size( MPI_COMM_WORLD, np )
        status = MPI_Status()

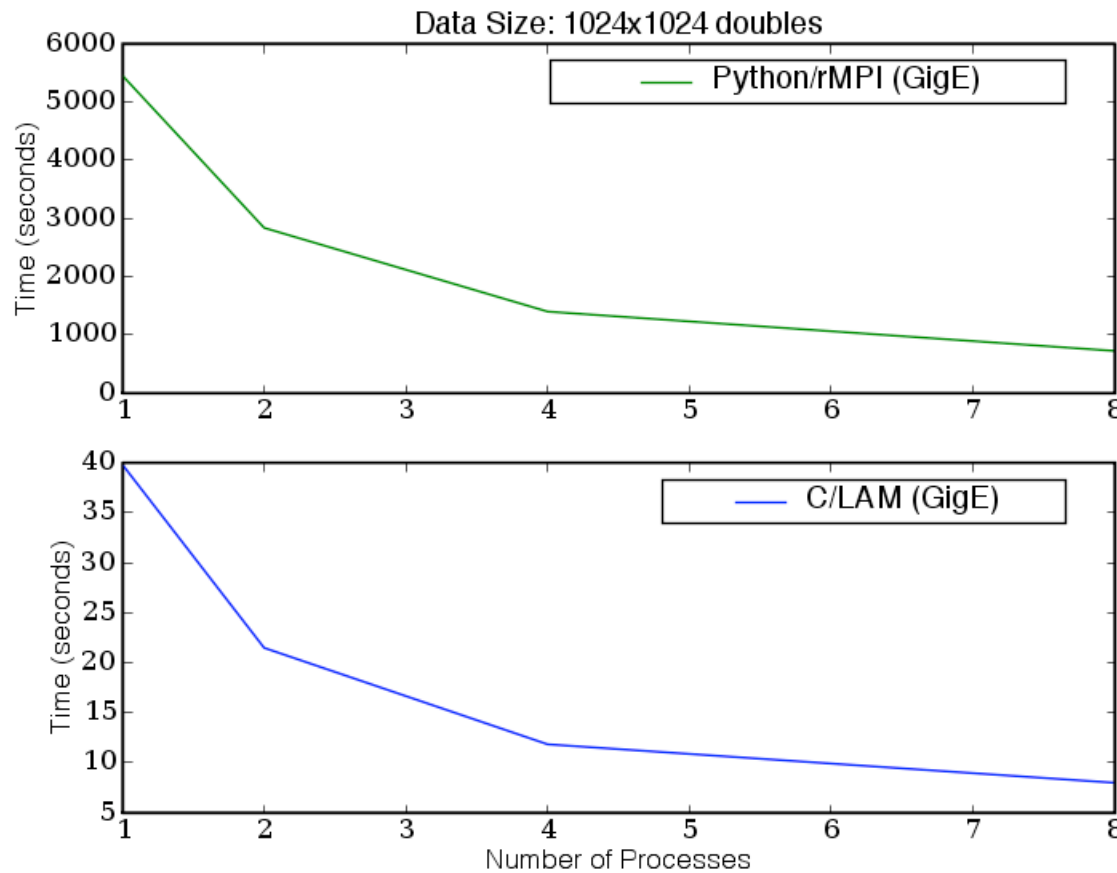
        recvbuf = [ 0.0 ]
        sendbuf = [ rank.value * 100.0 ]

        print 'Hello from rank %d' % ( rank.value )

        if rank.value == 0:
            for i in xrange( 1, np.value ):
                self.MPI_Recv( recvbuf, 1, MPI_FLOAT, i, 0, MPI_COMM_WORLD, status )
                print 'From rank %d: %f' % ( i, recvbuf[0] )
        else:
            # if not rank 0, send value to rank 0
            print 'Rank %d sending %f' % ( rank.value, sendbuf[0] )
            self.MPI_Send( sendbuf, 1, MPI_FLOAT, 0, 0, MPI_COMM_WORLD )

        self.MPI_Finalize()
```

# rMPI Conjugate Gradient



Penguin Cluster

AMD Opterons  
(Dual dual-core) 2.0GHz  
4 GB RAM, GigE

One VM/Process per node

# Dissemination Barrier

---

```
def MPI_Barrier( self, comm ):
```

```
    root = mpi_Rank(0)
    sendbuf = [1]; recvbuf = [0]
    msgUid = self.getMsgUid()
    status = MPI_Status()
```

```
    i = self.rank.value
    p = comm.size()
```

```
    steps = int(math.ceil(math.log(p, 2)))
```

```
    for k in xrange(steps):
```

```
        dest = (i + 2**k) %p
```

```
        src = (i - 2**k + p) %p
```

```
        self.MPI_Send(sendbuf, 1, MPI_INT, mpi_Rank(dest), msgUid, comm)
```

```
        self.MPI_Recv(recvbuf, 1, MPI_INT, mpi_Rank(src), msgUid, comm, status)
```

```
    return MPI_SUCCESS
```

# Experience

---

- ▶ Trickle
  - ▶ Design: about 2 days
  - ▶ First implementation: about 1 evening
  - ▶ Refinements easy (dynamic scheduling)
- ▶ rMPI
  - ▶ First implementation: about 1 month
  - ▶ Used in a Grad parallel computing class

# Contributions

---

- ▶ Python framework targeted at building parallel run-time systems
- ▶ Python-based approach for full, transparent checkpointing and migration

# Related Work

---

- ▶ Python
  - ▶ pypmi, mympi, pyro, twisted, others
  - ▶ IPython (Trickle-like functionality)
- ▶ VM level checkpointing and migration
  - ▶ Many Java-based implementations

# Future Work

---

- ▶ Refine the River core
- ▶ Further experimentation with rMPI and alternate implementation
- ▶ Evaluate different checkpointing schemes
- ▶ Develop new extensions: GAS-like language
- ▶ Automate the translation of a River implementation into a C/Java implementation

# River Website and Release

---

<http://www.cs.usfca.edu/river>

- ▶ Coming soon
- ▶ Final drafts of
  - ▶ SFM Paper
  - ▶ Trickle Paper