DsageNg

Distributed
Parallel
Asynchronous

Collaborative

Sage

Outline

- What
- ~~Why~~
- How
- When

What

- Distributed
  - Sage ”Over the Wire”
- Parallel
  - So much to do, so little time
- Asynchronous
  - Everything is an event to be ”handled”
- Collaborative
  - A short walk from Distributed, Parallel, Asynchronous

Distrubuted Computing Deconstructed

- Processing
- State Management
- Communications
- Programming

cpu gpu memoryHierarhchy
myranet gig-e threads
processes multicore lan wan
fpga persistance raid
asynchronous synchronous
coupling interaction bigIron
cluster multiCluster
faultTolerant replication security
heterogeneous multiUser
computeBound ioBound

Levels

High Level

- Management
  - Instantiation
  - Initialization
  - Control / Termination
- Scatter / Gather
- User Interaction

Low Level

1) Performance
2) See 1
3)
4)
5) Ease

Problem Decomposition
From Sage's Perspective

Accomplish

- Speed
- Exogenous Event Handling
- Collaboration

Considering

- Latency
- Bandwidth
- Decomposition
- Cost / Cycle
- Endogenous Events
- Heterogeneity

Considerations Expanded

Transports

- Shared Memory
- LAN
  - Infiniband/Myranet...
- WAN
  - Security

Programming

- Processors
- Threads, Processes, Coroutines
- Asynchronisity

Asynchronisity

- Everything is an Event
  - Synchronous
  - Asynchronous

Threads are Evil!!!

(Long Live Threads)

- Programming Model
  - Polling (stupid)
  - Blocking
    - Hold that thought
  - Coroutines
    - Good with issues
  - Callbacks

Blocking

- Application Level code shouldn't block
  - We have ”things” for that
- Applications block because the application programmer doesn't know how to do it any other way
  - Not entirely true

Twisted

- Framework for building Asynchronous Systems
  - Everything is asynchronous....
- Whats Important?

1) Reactor
2) Deferreds
3) Things built on 1) and 2)

Reactor

- Select on steroids
- Blocking done right
- Event Dispatching
  - Timed
  - Exogenous

Callbacks

- Continuous
  - eventSource.registerCallback(fn1, *args, **kargs)
  - def fn1(arg1, arg2, *args, **kargs):
    - ...
- One Time – use Deferreds (D)
  - D = object.doSomething(*inArgs, **inKargs)
  - D.addCallback(fn2, *args, **kargs)
  - def fn2(res, *args, **kargs)
  - ...

Deferreds

- Deferred – A Promise
- An object (defer.Deferred) with two important methods
  - D.addCallback(...)
  - D.callback(...)
    - This can be called multiple times = CallbackChain

```
42 def main():
43     def gotResult(result):
44             print "result is: ", result
45     d = O.doLongCalculation()
46     d.addCallback(gotResult)
47     d.addCallback(lambda r: reactor.stop())  # throw away the result
48     return d
```

What Code Looks Like

```
d = O.doComplicatedLengthyCalc(a, b, c) # returns deferred
def gotResult(result)
    print "important message: ", result
    return "a more important message"
d.addCallback(gotResult)            # first callback
def gotResult2(result, i2, s2, specialLabel)
     print "gotResult2 returns: ", result # will print "a more important message"
     print i2, s2, specialLabel             # will print "2, string2, special"
d.addCallback(gotResult2, 2, "string2", specialLabel="special")
d.addErrback(errfn,...)    # exception handling for deferreds
```

A deferred can return a deferred

If you need to call a method that returns a deferred within your callback chain, just return that deferred, and the result of the secondary deferred's processing chain will become the result that gets passed to the next callback of the primary deferreds processing chain

Twisted Architecture