University of Reading

Toward Scalable, Extensible Service Mashups: adding computing power to e-Science

Carl Albing

Cray, Inc. and University of Reading

artwork © University of Reading 2008
content © Carl Albing 2008

University of Reading

Today

- intro/motivation
- requirements
- software solution - DNWay
- enabling technology - Tycho
- example use - Monte Carlo PI
- future directions

Note: this work is done as part of my studies at the ACET/PEDAL lab and is not intended as a description, formal or informal, of features or products of Cray, Inc.

Intro/Motivation

University of Reading

- Peta-scale Supercomputing
  - 10,000 processors
  - 20,000 - 40,000 cores
  - application launch
  - application failure...

Post-mortem cleanup

- Need to hear from every node
- Parallel execution
- Parameterized tasks
- Is this more widely useful?

Use for e-Science?

University of Reading

- Distributed Data / Mapping Application
  - e.g. Godiva2
  - data characterization of very large data sets
    - dynamic choice of data sets
    - non-local data scattered across the web
    - compute/gather statistics
    - subsets of the original data
    - run while other work was being done
    - fast results or not at all

Classic Approach

University of Reading

- Condor – MW
  - based on condor:
    - typical condor use doesn't fit this model
    - MW (master/worker) extension
  - more setup required
  - re-request workers under threshold
  - message passing via:
    - Condor-PVM or
    - MW-File a file-based, remote I/O scheme
  - “local” only; can't cross firewalls

Requirements and Challenges

University of Reading

- Immediate access to compute cycles - not queued
- Robust - workers can come and go.
- Adapt to changing response times.
- Using processors and networks of varying speed.
- Simple interface; small number of methods.
- No configuration files.
- Limited to idempotent computations

The developer wishing to use this need only focus on defining the chunks of work, or “work units”, that will be distributed, and then undertaking that work when such a unit is delivered.

University of Reading

DNway

- Distributed *n*-way branching of execution
- follows the master/worker model
- master
  - provides work units
- workers
  - produce results for a work unit

DNway

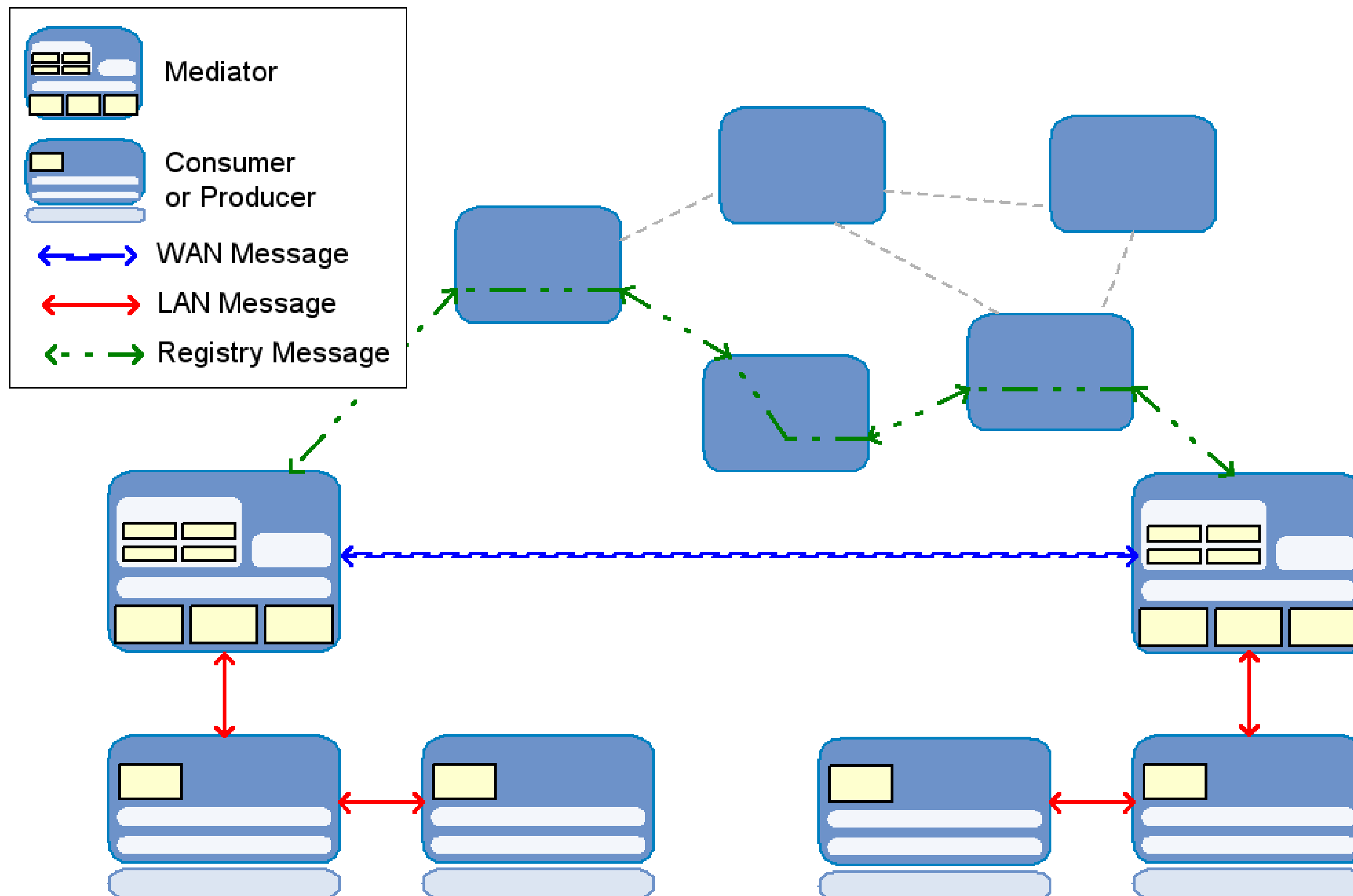
University of Reading

- simple API
  - only two classes
- uses Tycho
  - for distributed registry
  - for message passing
- hides the Tycho calls
  - workers register
  - master queries for workers
  - master sends work units to workers
    - resupplies with new work
  - master manages timeouts, retries

Enabling Technology

University of Reading

- Tycho



DNway Resiliency

University of Reading

- Startup
  - Work is sent as Tycho queries respond
- Ongoing
  - Work is sent as work is completed
  - Natural load balancing:
    - faster machines/connections get more work
    - slower machines get less work
- On timeout, work is either
  - Sent elsewhere
  - Done locally
  - Requirement: idempotent actions
  - Expired worker is ignored thereafter

Defining Work Units

University of Reading

Implement these methods for the master:

- These four are trivial:
  - getRetryScheme() -- just return a 1 or 2
  - getTimeoutValue() -- return # of seconds
  - byte[] getWorkCode() -- usually NULL
  - getWorkerID() -- return a fixed string

These two are the "meat"

- getWorkIterator()
  - "work unit" is returned by iterator
- doResults(result)

Doing the Work

University of Reading

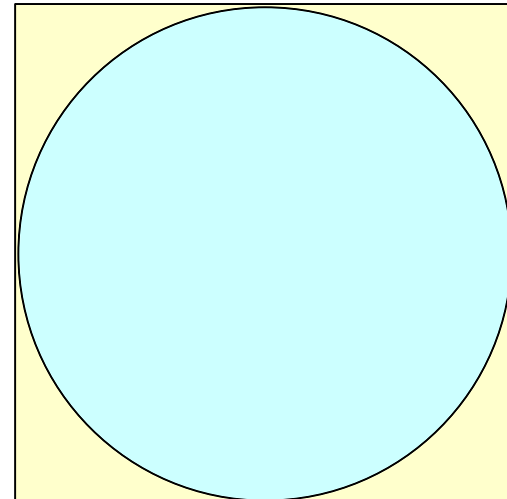
Implement these methods for the worker:

- getWorkerLabel() - same label as above
- doWork(workToDo) - the “guts” of it
- getResult() - package it up as a String

DNway Example: PI

University of Reading

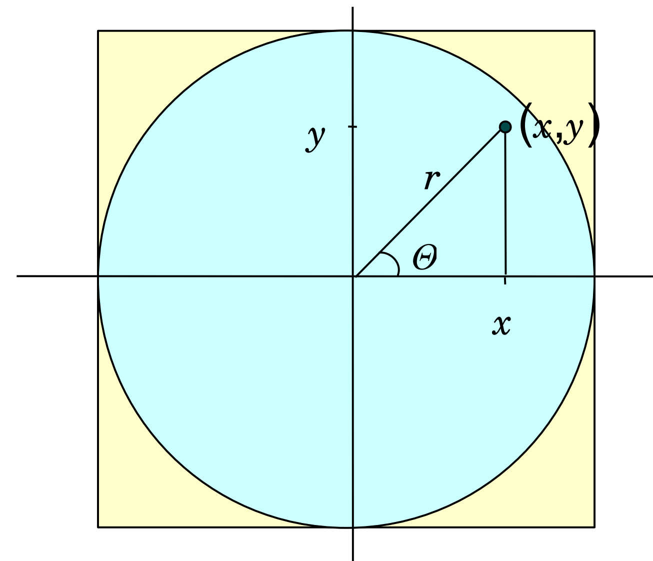
- Monte Carlo approximation of Π
- Random points in Unit Square
- Some fall in the Unit Circle
- Ratio is an approx. for PI/4
- We will create two classes
  - Monte: master
  - Carlos: worker(s)



Carlos: the Worker

University of Reading

- getWorkerLabel()
  - just return a fixed string, e.g. “Carlos”
- doWork(Object workToDo)
  - work unit: # of random #s to generate
  - for each #, pick (x,y), convert to (r,Θ)
  - remember Pythagoras?
    - $c^2=a^2+b^2$
    - $r=sqrt(x^2+y^2)$
  - if r < 1 then count it.
- getResult()
  - return the count



Monte: the Master

University of Reading

- getRetryScheme() returns 1 (DIY)
- getTimeoutValue() returns 5
- byte[] getWorkCode() returns null
- String getWorkerID() returns “Carlos”
- java.util.Iterator getWorkIterator()
  - returns an iterator
  - 1,000,000 100 => iterates over 100 objects
  - each is “10000”
- void doResults(Object result)

Sample Results

University of Reading

```
...
pi ~= 2.9530796
Sending work unit [100000]
Got a reply msg from socket://localhost:15803/producer/Carlos-swiss.us.cray.com
[78512]
pi ~= 2.9844844
Sending work unit [100000]
Got a reply msg from http://sinkhole:8080/?1704926223
[78523]
pi ~= 3.0158936
Sending work unit [100000]
Got a reply msg from socket://localhost:15803/producer/Carlos-swiss.us.cray.com
[78535]
pi ~= 3.0473076
Sending work unit [100000]
Got a reply msg from socket://localhost:15803/producer/Carlos-swiss.us.cray.com
[78620]
pi ~= 3.0787556
Sending work unit [100000]
Got a reply msg from socket://localhost:15803/producer/Carlos-swiss.us.cray.com
[78451]
pi ~= 3.110136
No more work for [socket://localhost:15803/producer/Carlos-swiss.us.cray.com].
Got a reply msg from http://sinkhole:8080/?1704926223
[78435]
pi ~= 3.14151
No more work for [http://sinkhole:8080/?1704926223].
$
```

Carlos – code:

University of Reading

```
public class Carlos extends DNwayWorker
{
    private int count;
    private Random gen;

    String getWorkerLabel() { return "Carlos"; }
    String getResult() { return count+""; }
    void
    doWork(Object workToDo)
    {
        double x; double y; int imax;

        if (gen == null) { gen = new Random(); }
        imax = Integer.parseInt((String)workToDo);

        count=0;
        for (int i = 0; i < imax; i++) {
          // generate 2 random numbers that lie inside the square;
          // centered at (0.0, 0.0) of radius 1.0
          x = 2*gen.nextDouble()-1.0;
          y = 2*gen.nextDouble()-1.0;
          // if (xcoord, ycoord) lies inside the circle
          if (java.lang.Math.sqrt(x*x+y*y) < 1.0) {
              count++;
          }
        }
    } // doWork

    public static void
    main(String[] args)
    {
        Carlos producer = new Carlos();
        producer.register();
    }
} // class Carlos
```

Monte - code:

University of Reading

```
public class Monte extends DNwayDistributor
{
    private int maxSize, npieces;
    private double rtot, piapprox;

    int getRetryScheme() { return 1; }
    int getTimeoutValue() { return 5; }
    byte[] getWorkCode() { return null; }
    String getWorkerID() { return "Carlos"; }
    Iterator getWorkIterator() { return new NumChunks(maxSize, npieces); }

    void
    doResults(Object result)
    {
        rtot += (double) Integer.parseInt((String)result);
        piapprox = 4.0 * rtot/maxSize;
        System.out.println("pi ~= "+piapprox);
    }

    Monte() { } // constructor
    Monte(String max, String nparts)
    {
        this();
        maxSize = Integer.parseInt(max);
        npieces = Integer.parseInt(nparts);
        rtot = 0;
    } // constructor

    public static void
    main(String[] args)
    {
        Monte consumer;
        if (args.length == 2) { consumer = new Monte(args[0], args[1]); }
        else { consumer = new Monte("1000", "10"); }
        consumer.register();
    }
} // class Monte
```

Use for e-Science

- First approach
  - Design assumption:
    - data file(s) local to the distributor
    - data sent to workers via Tycho.
- Distributor:
  - reads, sends chunks of data as it is read,
  - sends these as work units to each worker.

Use for e-Science

- Second approach
  - Design assumption:
    - data remote from the distributor
    - data available over n/w to workers.
- Distributor:
  - Define a "work unit" as
    - a URL
    - an offset and
    - a length

University of Reading

Future Work

- Use in an e-Science application
- Empirical comparisons with more traditional approaches.
- Both Java and C/C++ language interfaces.
- Sending worker code to generic worker.
- Extended architectures
  - multiple workers per node
  - recursive hierarchy

Acknowledgments

University of Reading

- Prof. Mark Baker, supervisor
- Dr. Mat Grove - Tycho
- Dr. Jon Blower - Tech. Dir. Reading e-Science Centre