Using Jini and JavaSpaces for Clustered 3D Rendering

Simon Kent - Brunel University

Nigel Warren - Zink Digital Ltd

Brunel UNIVERSITY WEST LONDON

Introduction

- Rendering problem space
  - Bandwidth heavy
  - Computationally heavy
  - Distributed
- Aim
  - Improve on current approaches
  - Understanding capabilities of Jini and JavaSpaces in this environment

Brunel
UNIVERSITY
WEST LONDON

Rendering Process

Scene Descriptor
(RIB File)

Renderer

Rendered Images

Brunel
UNIVERSITY
WEST LONDON

Angel Renderer

- Developed by Dr Ian Stephenson at NCCA
- Pixar Renderman Interface Compatible
  - RIBs and shaders
- Portable ‘C’ code (not open source)
  - Windows, Mac OSX, Solaris, Linux, SGI
- Two pass rendering for Lighting calculations
  - >1 RIB file, >1 image file

Brunel UNIVERSITY WEST LONDON

Batch Renderers

Rendering is extremely computationally expensive so …

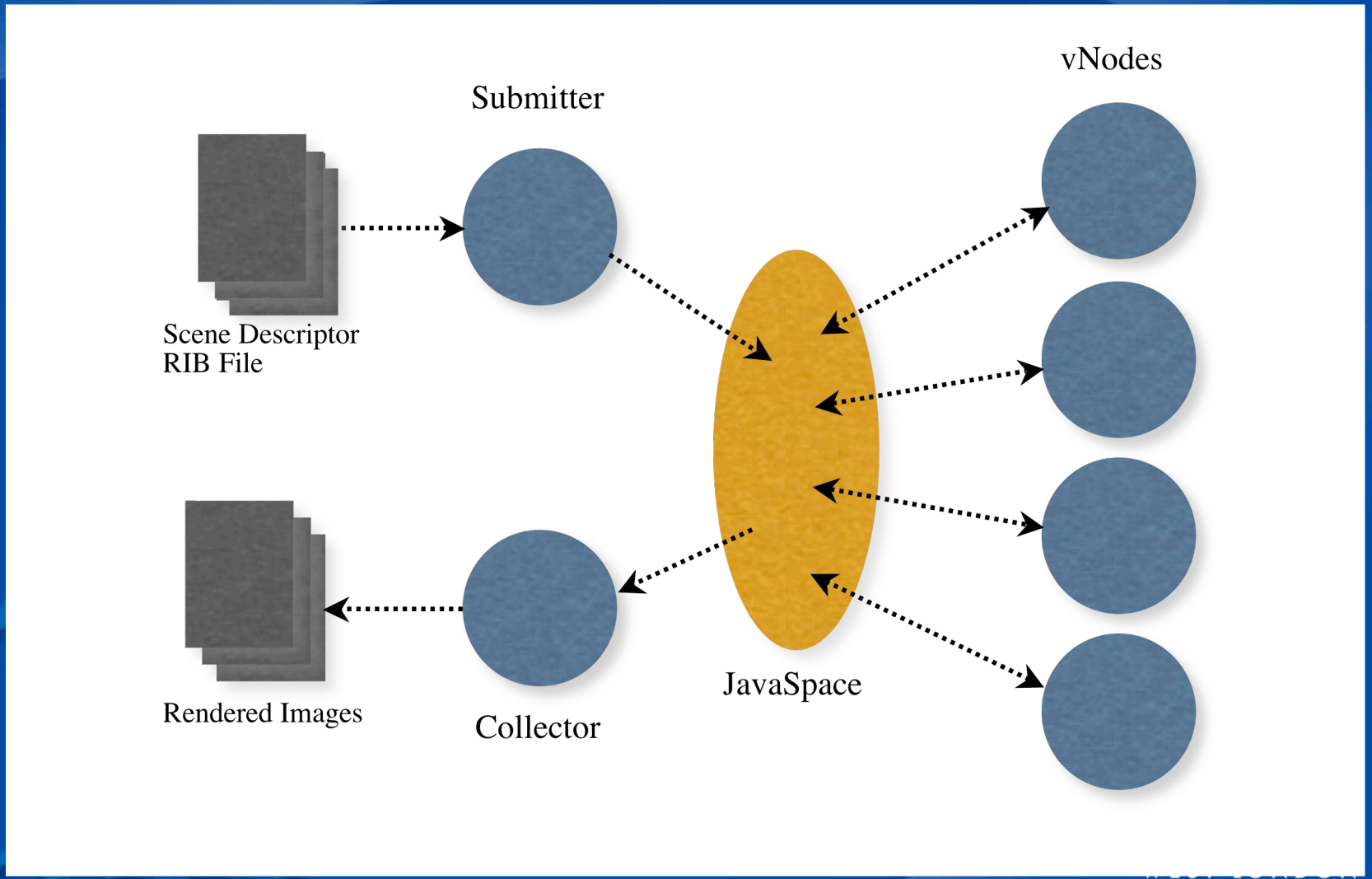
- Spread the work of rendering a number of frames over a set of machines and/or over time.
- Long running - Overnight, to a number of days
- Normally implemented by scripts that generate scripts that embed IP addresses etc.
- Makes rendering ‘feature length’ movies possible

Batch Renderers

Batch renderers can suffer from a mix of issues:

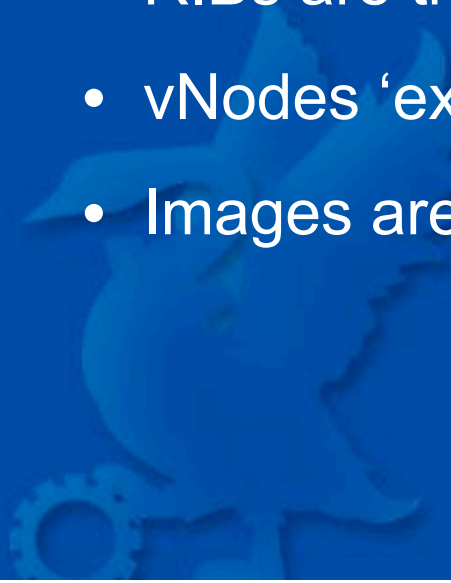
- Tight coupling of batch controller to workers
- No ‘dynamic’ changes (can run for days)
- Long running single point-of-failure for batch controller
- Failure of render job due to worker failure

Designing Spray



Designing Spray

- Used Jini for service discovery and lookup
- Used Blitz JavaSpaces
  - Configured for large entries - RIBs & Images
- We built Angel installs for each host platform
- RIBs are transported as JavaSpace Entries
- vNodes ‘exec()’ to Angel on the host machine
- Images are also sent back as Entries



Brunel
UNIVERSITY
WEST LONDON

Using exec()

- Use sys prop “file.separator”
- Use Runtime.getRuntime().exec(…) to make a java.lang.Process
- Use threaded stream handlers for the
  - Input Stream
  - Error Stream
- Use java.lang.Process to control the exec’d process.
  - E.g. java.lang.Process.waitFor()

Test Platforms

- Windows - on AMD and Intel
  - 1 & 2 processors
- Mac OSX - on G4, G5 and Intel
  - 1, 2 & 4 processors
- Linux - on Intel and AMD
  - 1 & 2 processors
- Solaris 10 - on Sparc 64
  - Single processor blades

Brunel UNIVERSITY WEST LONDON

Production Systems

- Mac Quad G5
  - 4 processors, 2.5 GB
- Class 2 Beowulf Cluster
  - 24 Rack nodes of 2 P4 processors
  - Centos 4.1
- The Gridlet (Grid ‘On Tour’)

  7 - Sun Netras and Sun V100s

Brunel UNIVERSITY WEST LONDON

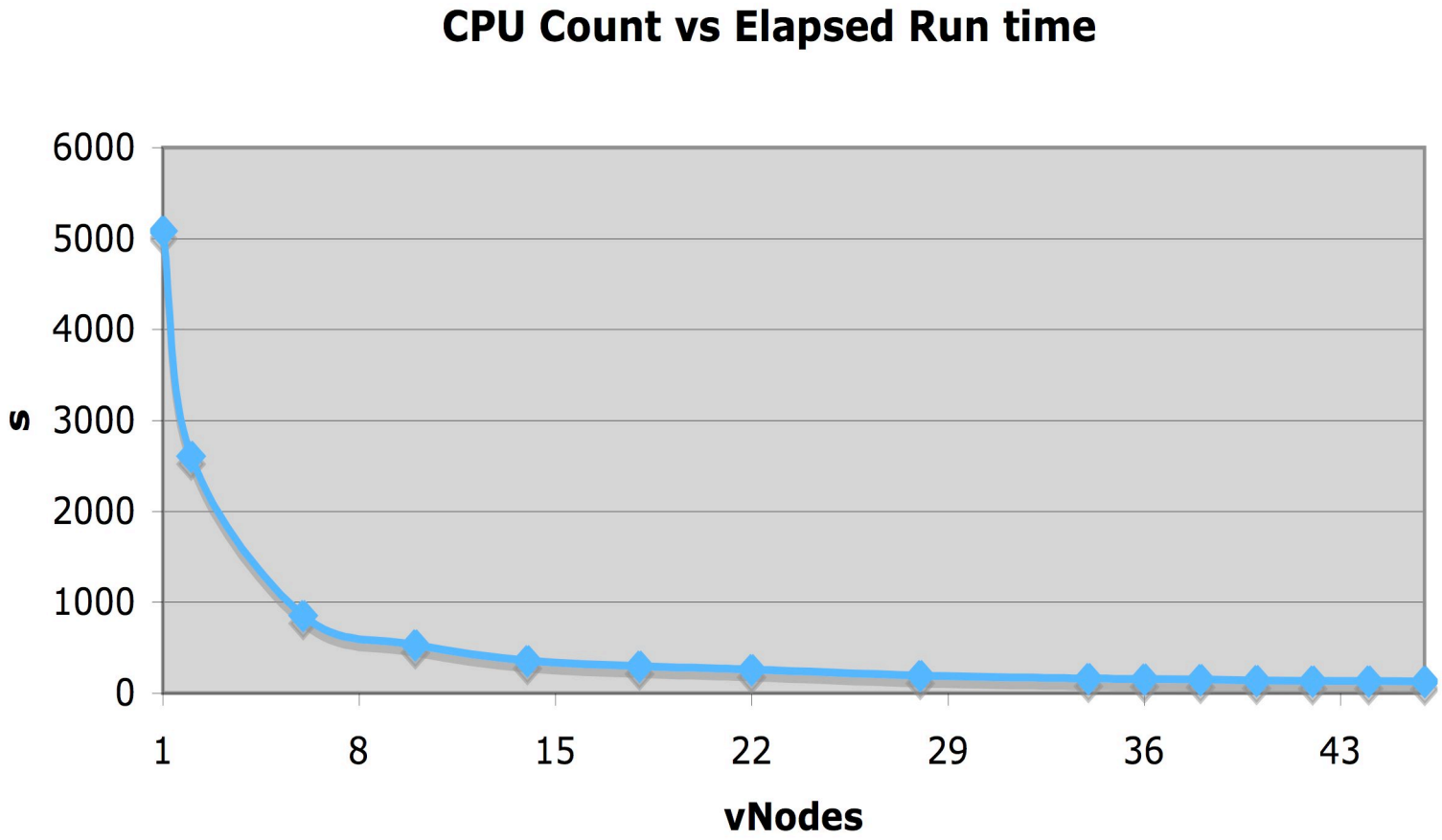
Test Movies

Newell Teapot - “hello world” of rendering

- 360 HD Frames (120kB/frame)
- 1- 46 vNodes on Beowulf Cluster
- 100Mbps network

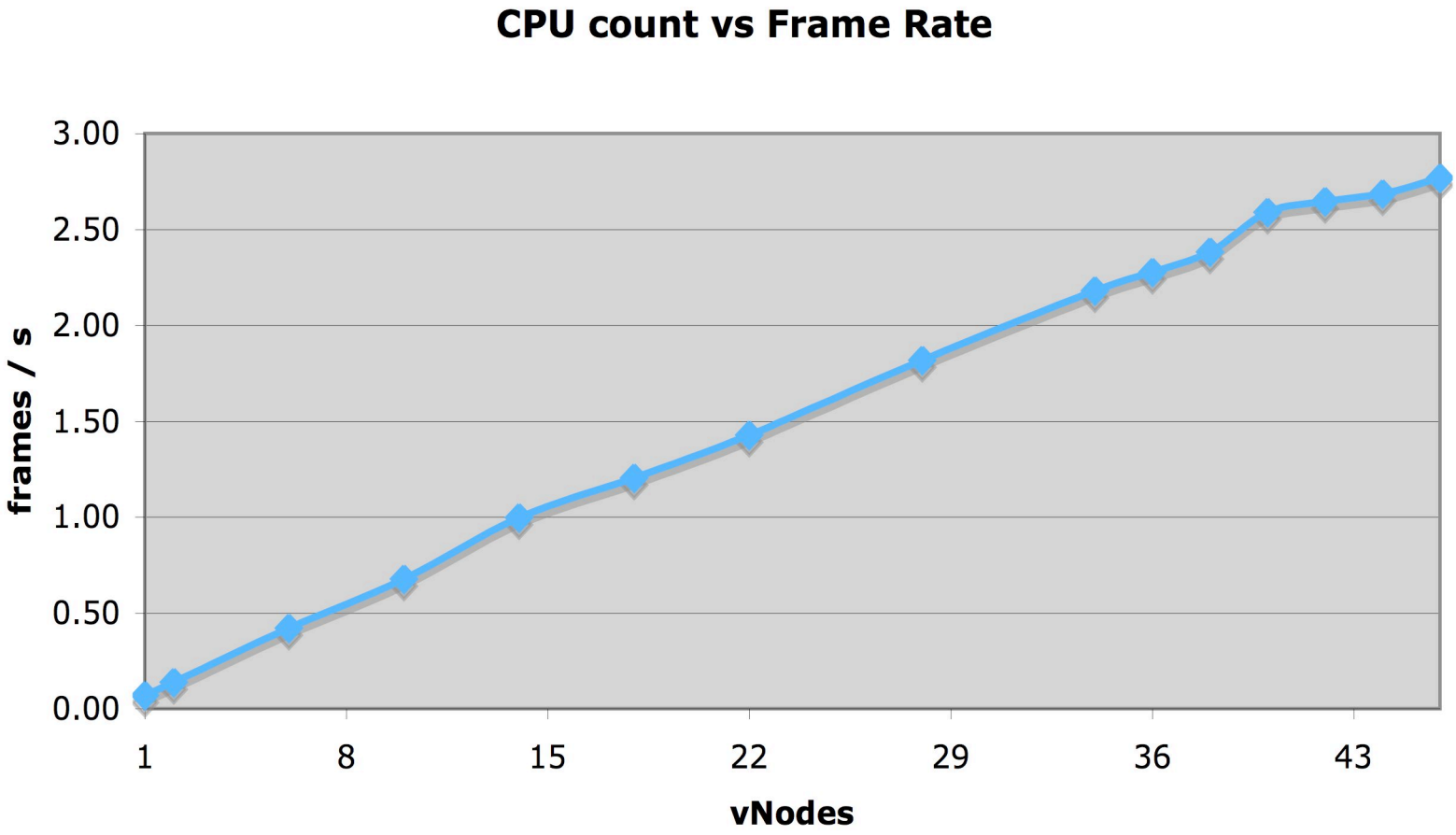
Brunel
UNIVERSITY
WEST LONDON

Run Time



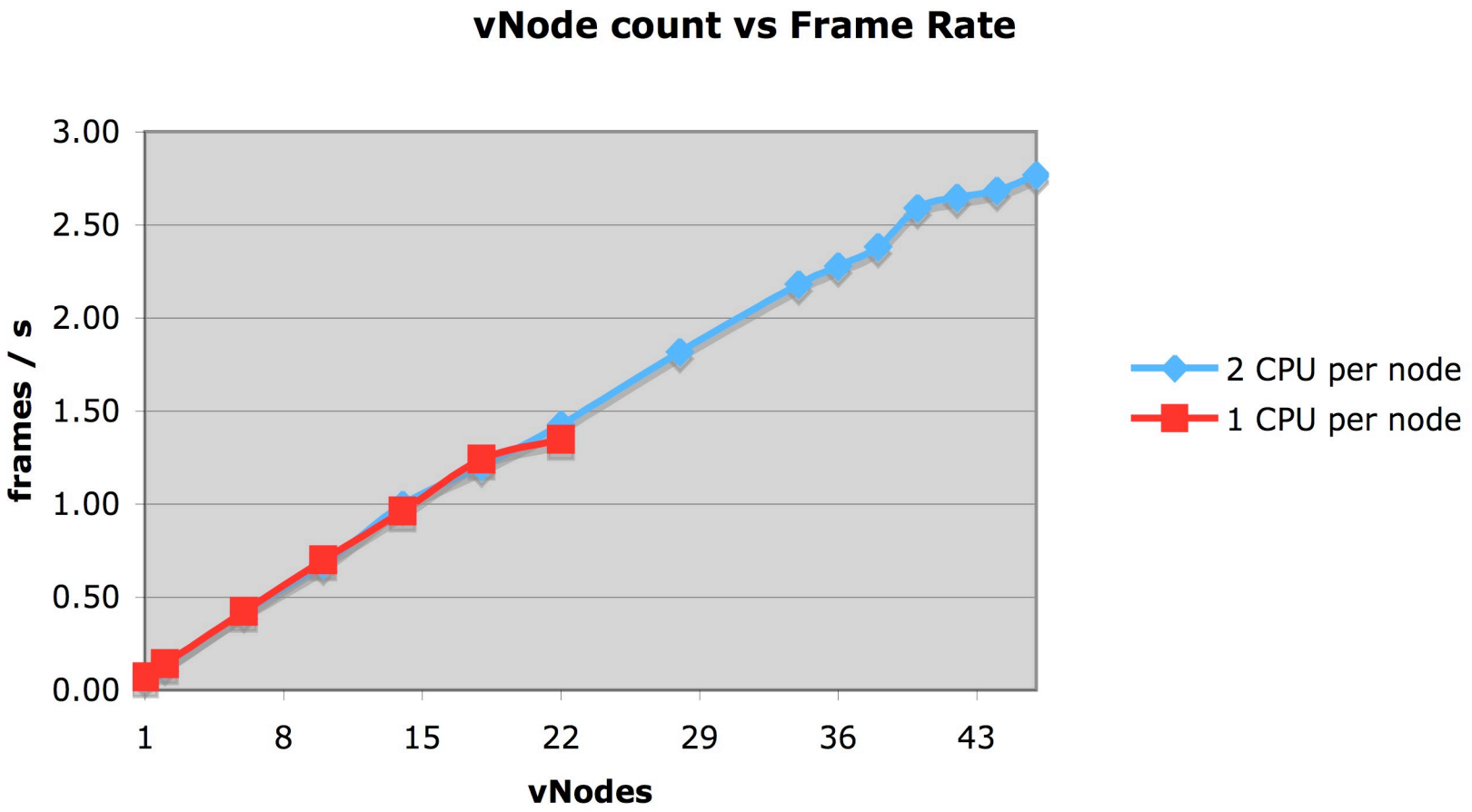
Brunel UNIVERSITY WEST LONDON

Frame Rate



Brunel UNIVERSITY WEST LONDON

One vs Two processor



Brunel UNIVERSITY WEST LONDON

Conclusions

- Easy to combine
  - Distribution and fault tolerance of Jini platform
  - High performance native platform code
- Transport files over JavaSpaces
- Scalable to realtime rendering possible
  - Some realtime SD processing done

Brunel UNIVERSITY WEST LONDON

Future work

- Test assumptions regarding network bandwidth.
  - Optimistic about scalability
- Test with production quality RIB files
- Generalise infrastructure for other distributed processing tasks

Brunel UNIVERSITY WEST LONDON

Thank you for Listening

Brunel
UNIVERSITY
WEST LONDON