

---

# What Does Hacking Have to Say About Building Adaptive, Autonomous Grid Programs?

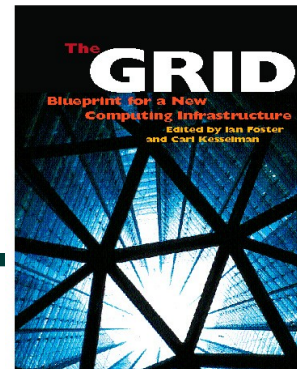
**Rich Wolski**

**Computer Science Department  
University of California, Santa Barbara**

# The Grid Paradigm

---

- **Vision: Application programs “plug” into the system to draw computational “power” from a dynamically changing pool of resources.**
  - **Electrical Power Grid analogy**
    - **Power generation facilities == computers, networks, storage devices, palm tops, databases, libraries, etc.**
    - **Household appliances == application programs**
- **Scale to national and international levels**
- **Grid users (both power producers and application consumers) can join and leave the Grid at will.**
- **Secure, Reliable, Cheap, Fast...**
- **At what level of abstraction should it be implemented?**
  - **System approach**
    - **Make it look like a single computer of some sort**
  - **Application-level approach**

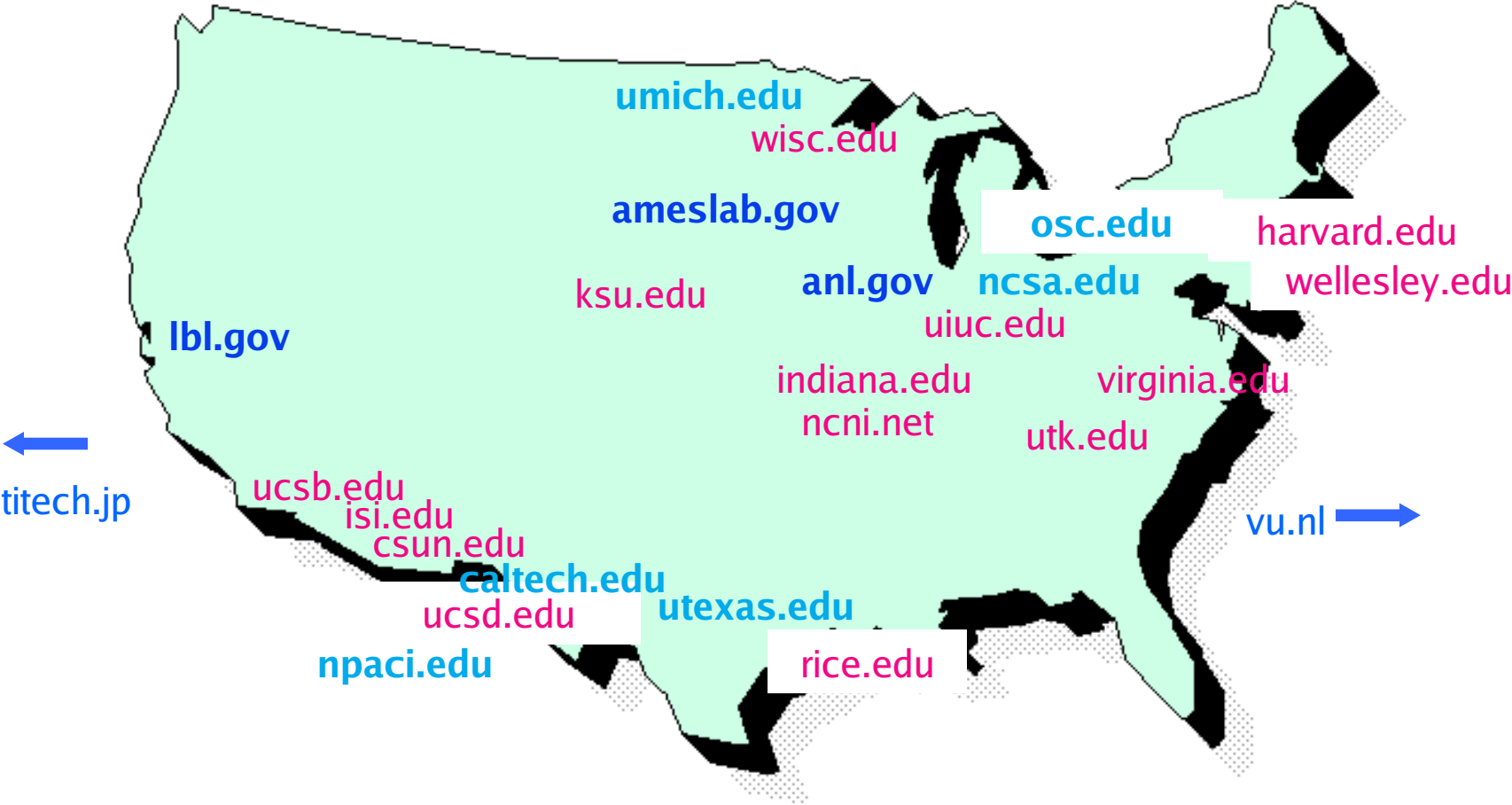


# Why Implement at the Application Level?

---

- **Computer resource occupancy epochs are not electrons**
  - The “power” delivered by a resource to an application depends on the application
    - It is difficult for the system to do the adaptation and optimization
- **The application “knows” where it can be adapted, rescheduled, etc.**
  - Data structures within the application dictate how flexibly it can be deployed
  - Numerical and efficiency concerns
    - System is not free to shape the application’s execution at will
- **An application can only use a subset of the total resource pool**
  - The execution environment is not uniform across all resources
  - User access defines a maximal pool

# For Example: Rich's View of "The Grid"



# Programming Paradigm

---

- **If the grid model manifests at the application level, what structure should the application have?**
  - **Service overlay**
    - Each resource hosts a set of application-specific “services” which can be composed dynamically
  - **Distributed application control program integrated with the computational components**
    - Application implements its own customized runtime system
- **EveryWare: assembly language for building adaptive grid programs**
  - **Adaptive messaging layer**
    - Automatic timeout discovery
    - Asynchronous socket management
  - **Performance-specified resource discovery**
    - Application must translate
  - **State synchronization service**

# First Principles Grid Applications (Hacking)

---

- **Ramsey Search**
  - Branch and bound algorithm
  - First program to sustain supercomputing performance levels despite heavily fluctuating resource pool
  - Proved small results for symmetric Ramsey numbers  $R(5,5)$  and  $R(6,6)$
  - No big result (yet)
- **GridSAT**
  - Master/slave control, unstructured P2P data
  - Out performs best-known complete SAT solver in 2004, 2005, and 2006
  - First program to use batch-queue prediction to optimize performance
  - Solved 15 previously unsolved SAT problems
- **LDPC**
  - Lowest error rates yet investigated

# Commonalities

---

- **All three applications have achieved new domain science**
  - Many results in grid computing reproduce results from parallel computing using a grid infrastructure
- **All three applications were designed from “first principles” to be EveryWare applications**
  - MPI and bulk-synchronous application structure was generally too rigid
  - Each has a performance model that was engineered along with it
- **All three applications are rapidly adaptive and highly robust**
  - Ramsey search ran for seven month continuously => difficult to stop
    - **First computer fungus**
- **All three applications were difficult to develop and even more difficult to debug**
  - The applications had to be multi-granular which made developing the algorithms themselves a huge challenge

# Some Observations

---

- **Some of the earliest domain science results have been produced by application-level systems**
  - Freed from legacy support
  - Designed specifically to exploit the characteristics of the grid
  - Results not possible without a grid
    - Not “classical” HPC results
- **Application specific service overlays are now coming into fashion**
  - Cloud computing, Amazon’s EC2, etc.
  - Virtualization level is much lower
  - Adaptivity and optimization problems are the same
  - So far, there has been no real investigation of programmability
    - Several start-ups are happening
- *Maybe the grid enables science in areas that parallel computing does not?*

# Observations about This Approach

---

- A grid program really consists of three phases
  - Compilation/program preparation
  - Deployment
  - Execution
- These phases may interleave
  - Compilation may occur during execution => JIT
    - C source code is, by far, the most highly portable intermediate form (with autoconf) for Linux/ Unix systems
  - Deployment may occur during execution => new resources become available
- Suggestion: any grid programming language should include first class abstractions for coding all three phases
  - Grid runtime services look very different
    - For example: Automatic, secure code repository

# Another Observation

---

- **Programs have an increasing number of hidden global variables => “the environment”**
  - Libraries (at appropriate revision levels)
  - Configuration (e.g. class path)
  - File system structure
  - Accounting information
- **Suggestion: a grid programming language should include abstractions for defining and manipulating “the environment.”**
  - Opinion: this support is not inheritance in the object-oriented sense of the word
  - Potentially far-reaching impact on the OS virtualization community
  - Serious performance ramifications
  - Provenance

# Grid Space-time

---

- **Grids are not flat in space**
  - Heterogeneous processor/machine pool
  - Non-uniform storage capabilities
  - Network topology has tremendous performance impact
- **Grid are not flat in time**
  - Best effort batch queues embody 7 orders of magnitude variation (measured in seconds)
  - Advanced reservations and co-allocation must be coordinated with file staging
- **Suggestion: to exploit heterogeneity, the programmer must be able to reason about the performance space in which the program executes.**
  - Language should include abstractions supporting this reasoning
    - Performance
    - Topology
    - Reliability

# One Final Observation

---

- **Scale is a tool for improving robustness**
  - Counter intuitive for HPC scientific programming community
    - **More components implies greater chance of single component failure**
- **Two approaches to exploiting scale for robustness**
  - Reactive
  - Replicative
- **If a component fails, scale maximizes the chance of finding a replacement**
  - Need a way to determine when something should be considered “failed” by the application
- **If failures are independent, replication allows rational composition**
  - Need a way to determine success/failure probability at the service level

# In Other Words

---

- A grid programming language should embody abstractions for
  - Program preparation
  - Deployment
  - Execution
- Program environment should be represented as first-class state
  - Clearly smaller is better
- Grid space-time topologies should be abstracted by the language
- All programming abstractions should include a way to express what it means for them to fail (failure semantics) and what should happen in response (remediation semantics)
  - E.g. automatic timeout discovery
- All programming abstractions should include a way to express success/failure probabilities
  - Ensuring independence will be a heavy burden on the runtime system

# Heresy or Truth?

- **These language requirements imply that grid programs are complex**
  - Application programmers must build portable, high-performance, robust, and adaptive services that can be deployed and manipulated under program control
  - MPI programming is trivial in comparison
- **What if grid programming is just hard (or just harder than parallel programming) at some level?**
  - It is clearly a new paradigm with added complexities
    - **Parallel programming is harder than sequential programming**
  - Important systems and applications tend to be complex and hard to develop
    - **Google, Netscape (browsers), iPhone, video games**
  - Several modern languages (for better or worse) take functionality over productivity or legacy as a mandate
    - **perl**
- ***Maybe we need a language that focuses on functionality first?***

# Thanks

---

- **Thanks**
  - NSF OCI, NSF NGS, VGrADS, SDSC, TACC, NCSA, Argonne, PSC
- **Middleware and Applications Yielding Heterogeneous Environments for Metacomputing**
  - Dan Nurmi, John Brevik, Graziano Obertelli, Matthew Allen (NWS and QBETS)
  - Wei Zhang, Sydney Pang (sensor networks)
  - Lamia Youseff, Shriram Rajagopalan, Dmitrii Zagordnov, Chris Grezgorczyk (EveryWare and Utility Computing)
  - Andrew Mutz (computational economies)
- **[rich@cs.ucsb.edu](mailto:rich@cs.ucsb.edu)**