Solving “Hard” Satisfiability Problems Using GridSAT

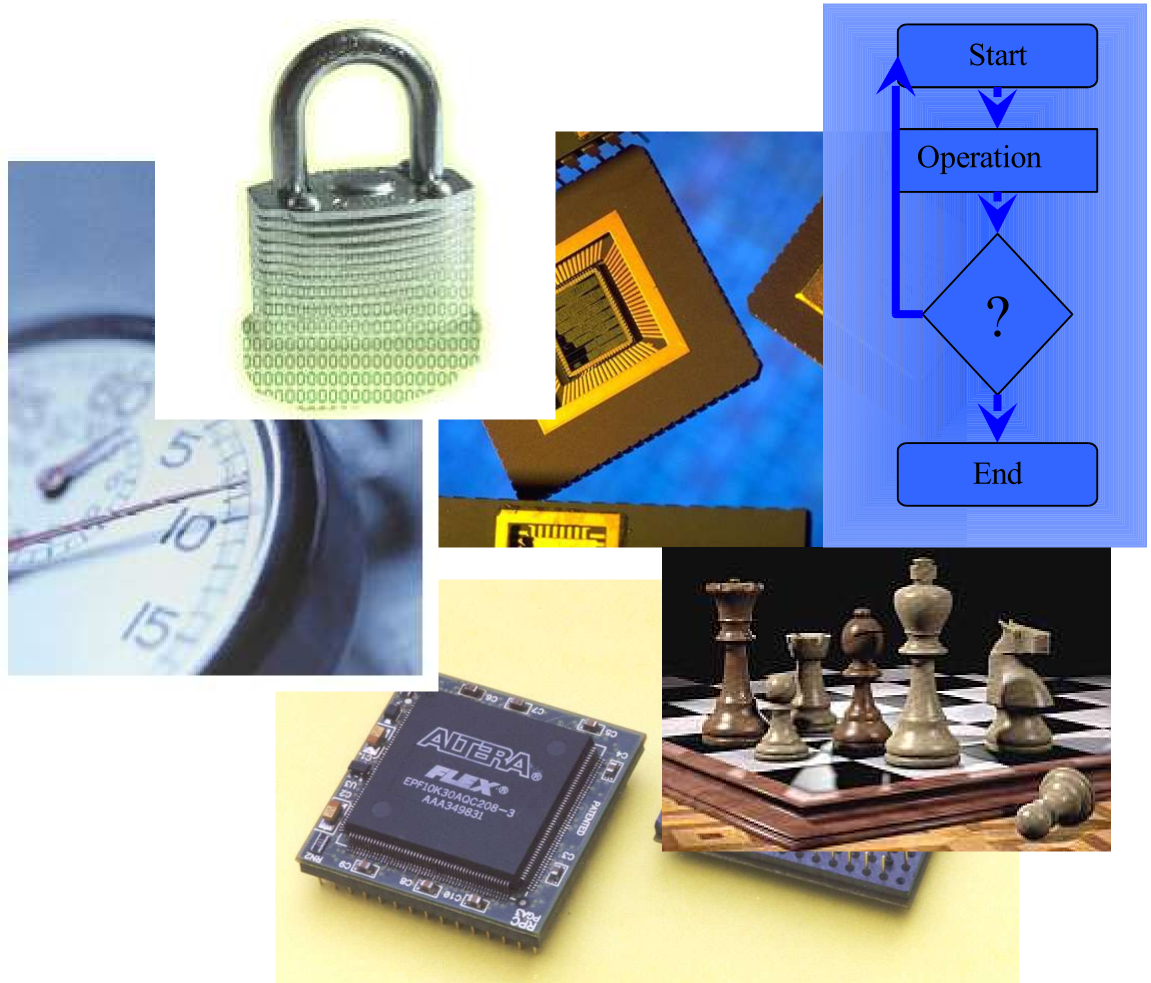
Wahid Chrabakh
And
Rich Wolski
University of California, Santa Barbara

Applications:

- Legacy applications:
  1. Embarrassingly Parallel: independent tasks
  2. Fixed granularity (mostly MPI): Communication and computation do not overlap
- New Applications:
  - Computation to communication ratio is adjustable
  - Communication and computation can overlap
  - Tasks cooperation: adjustable
- Suitable for a Computational Grid
- Tools:
  - Enable new applications using tools of grid computing
- Goal: Generate new domain science

Satisfiability Applications

- Circuit Design
- FPGA routing
- Model Checking:
  - AI, Scheduling
  - Software: OS
- Security
- Theoretical:
  - physics, chemistry, combinatorics
- Many More…



SAT Community

- SATLive: http://www.satlive.org/
  - News, forums, links, documents
- SATEx: http://www.lri.fr/~simon/satex
  - Experimentation and execution system
- SATLIB: http://www.satlib.org/
  - Dynamic set of Benchmarks
  - Freely available solvers
- Annual SAT competition
- Benchmark mostly from REAL applications

UCSB

Satisfiability Problem(SAT)

- Set of variables $V=\{v_i \mid i=1,\ldots,k\}$
- Literal: a variable or its complement
- Problems in CNF form: community standard
- Clause: OR of of literals
- Standard *CNF* File format:
  - $F=C_1C_2C_3\ldots C_k$
- Large search space: thousands, even millions of variables

p cnf num_vars num_clauses
c comments
+v1 –v2 … +v213 **0**

Sequential SAT Solver

- Explore and prune search space
- Learning for optimization: new clauses are deduced
- Resource Intensive:
  - CPU: heavy load + large search space
  - Memory resident clause database grows due to learning

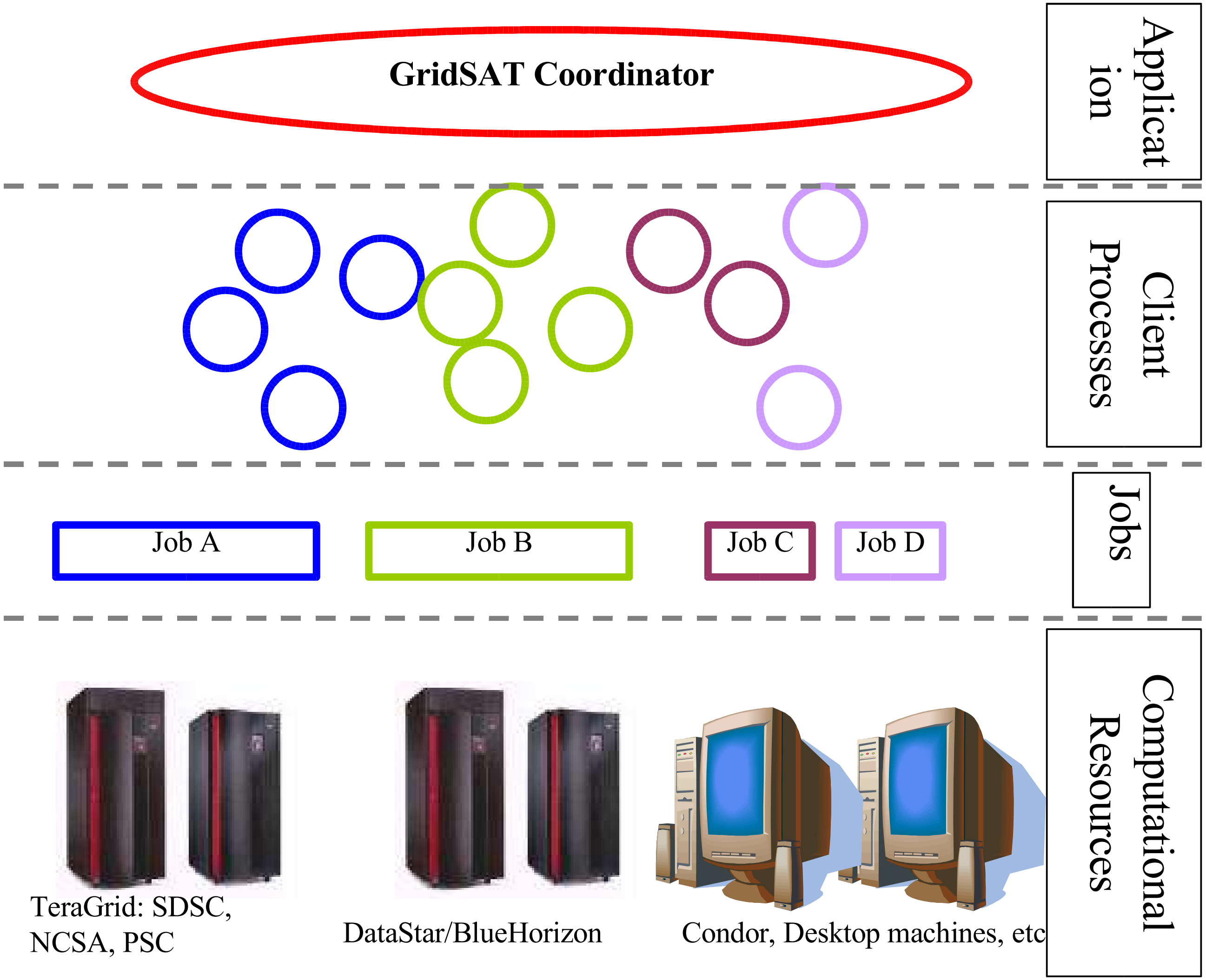
GridSAT: Parallel Solver

- Parallelizing of sequential solvers:
  - Allows using multiple resources
  - Splitting leading to a better performance
- Sharing learned clauses to improve performance:
  - Clause learned in one solver can be used by shared with other clients
  - Only important clauses are shared
  - Share clauses immediately

Scheduling Considerations:

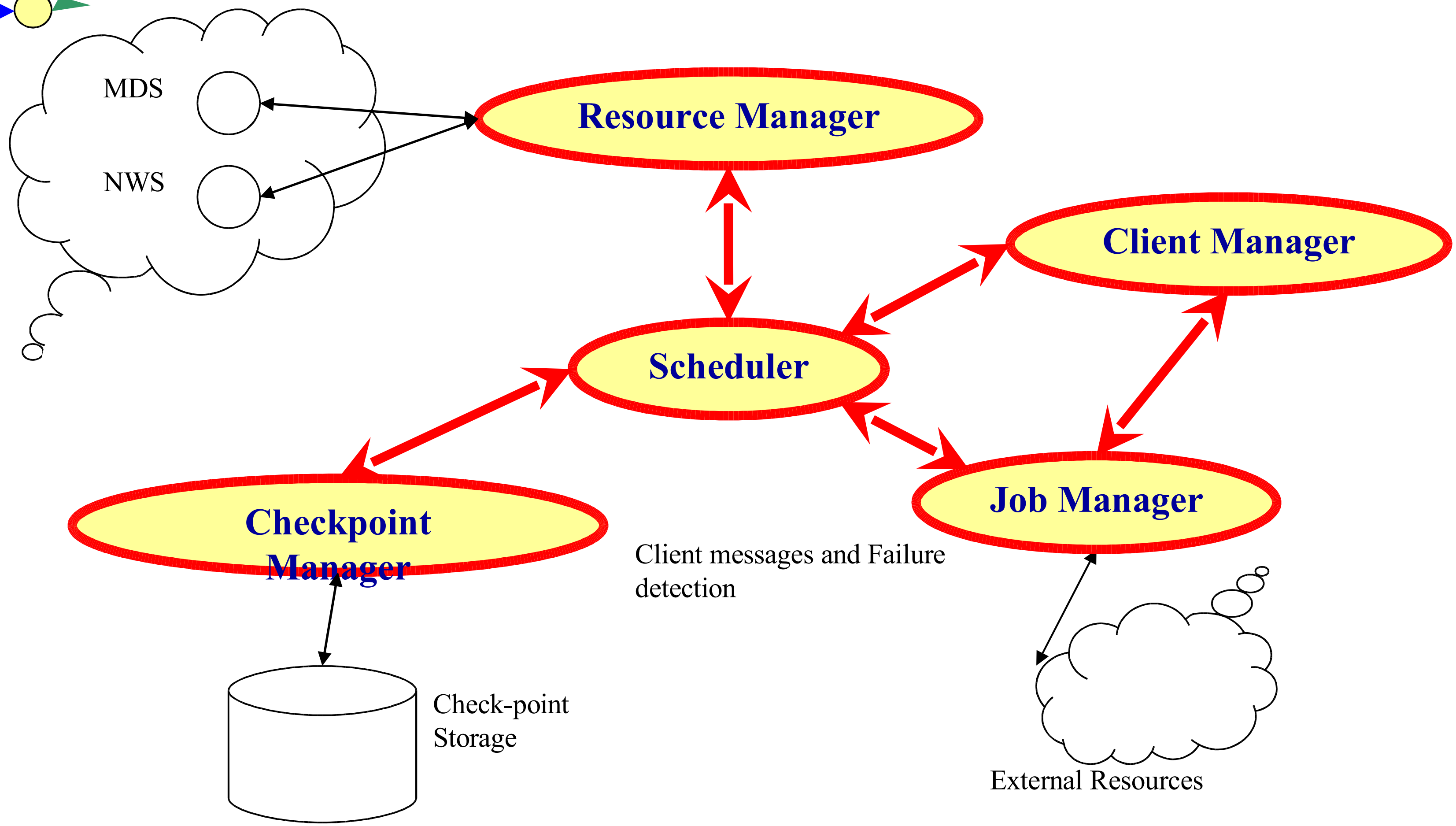
- Problems vary: easy to hard, short to long
- Cannot predict how difficult a problem is:
  - How much time it will take?
  - How many resources it will need?
  - When to split?
  - Which process splits first?
- Preferably adaptive:
  - Use few resources for "*easy*" problems,
  - More resources are gradually added for "*harder*" problems.

Resource Representation



UCSB

GridSAT Architecture:



Scheduler Design:

- Dynamic: Clients may join and leave at anytime
- Fault Tolerance:
  - Restart client in case of failure
  - Checkpointing: when significant part of the search space is eliminated
- Migration:
  - Migrate jobs to nodes with larger pool of neighbors
  - Decrease split and sharing overhead

UCSB

Resource Management:

- Start with one compute node
- Incrementally acquiring resources
- Aggressively Release resources:
  - When sub-problem is solved
- Resource usage expands and shrinks based on the problem behavior
- Work backlog: when all resources are busy,
  - clients which wish to split are kept in a queue
  - When resource is released or discovered it is assigned work by splitting or migration.

Scheduling Policies

- Timeout based:
  - Overall timeout: used with shared resources
- Budget Based:
  - For resources with wait-time: Batch controlled system
  - CPU count or timeout may not be fulfilled
  - Divide into multiple jobs
  - Use Max CPUs * Timeout as a budget
  - Debit from budget for every job
- Clients resource aware:
  - If memory is short on system: reduce memory use
    - Cooperative, OS penalizes processes with large memory

UCSB

Programming Requirements

- Dynamic Resource Pool
  - Application performance is best when minimal set used
- Error Handling:
  - Large number of resources & long execution
  - All resources fail: upgrades & maintenance
  - Timeout period for failure prone procedures
- Universal deployment:
  - Reduces development effort

UCSB

Implementation:

- Existing Technologies:
  - MPI-1 and 2: limited error handling,
  - Globus: callback functions, GRAM
  - WS: WS-Notification and WS-BasicFaults
- Previous version: uses GrADS
  - GrADSoft and testbed
- Current version: usesEveryWare:
  - Portable communication primitives with timeouts
- Resource management:
  - Use Globus if available, else use SSH
  - Application level implementation with timeouts

Experimental Setup

- Resources:
  - 40 machines from GrADS testbed: collection of UTK, UCSD and UCSB machines
  - TeraGrid: SDSC, NCSA
  - DataStar: SDSC
  - BlueHorizon: SDSC, retired

UCSB

Benchmark Problems

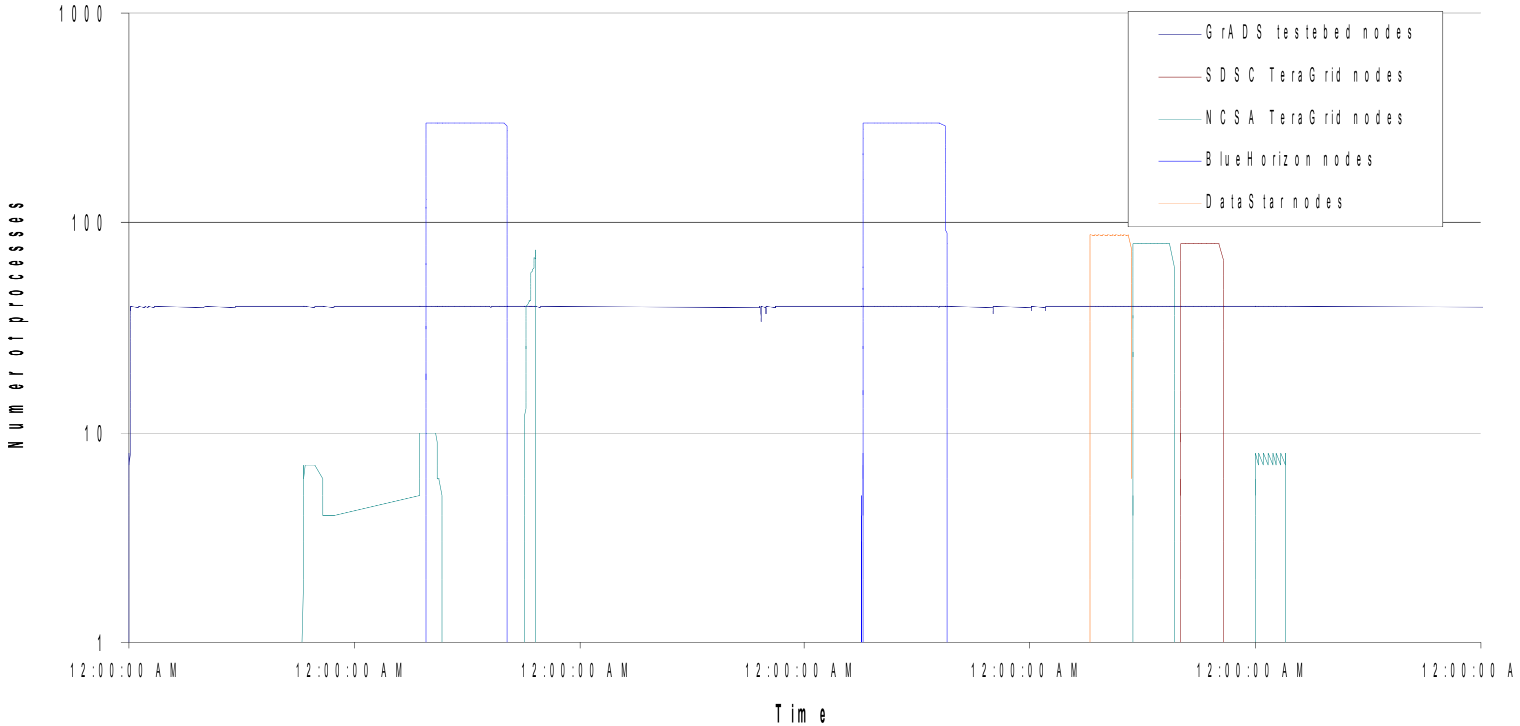
- Three categories:
  - Crafted, Random and Industrial Benchmark
- Challenging benchmark: not solved by any of the solvers
- All problems from challenging set:
  - FPGA, Model checking, theoretical
- Not solved with previous GridSAT version with testbed and BH

Results

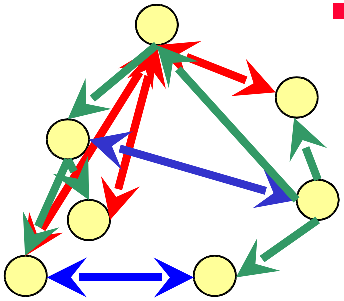
File name	Category	Solution	Time	Result
3bitadd-31	theoretical	UNSAT	8 days	__
k2fix-gr-rcs-w8	**FPGA**	*	**23 hours**	**UNSAT**
k2fix-gr-rcs-w9	**FPGA**	*	**14days+8hours**	**UNSAT**
cnt10	**theoretical**	**SAT**	**4 hours**	**SAT**
comb1	MC	*	11 days	___
f2clk50	MC	*	9 days	___
hanoi6	theoretical	SAT	23 days	___

***problem solution is unknown**

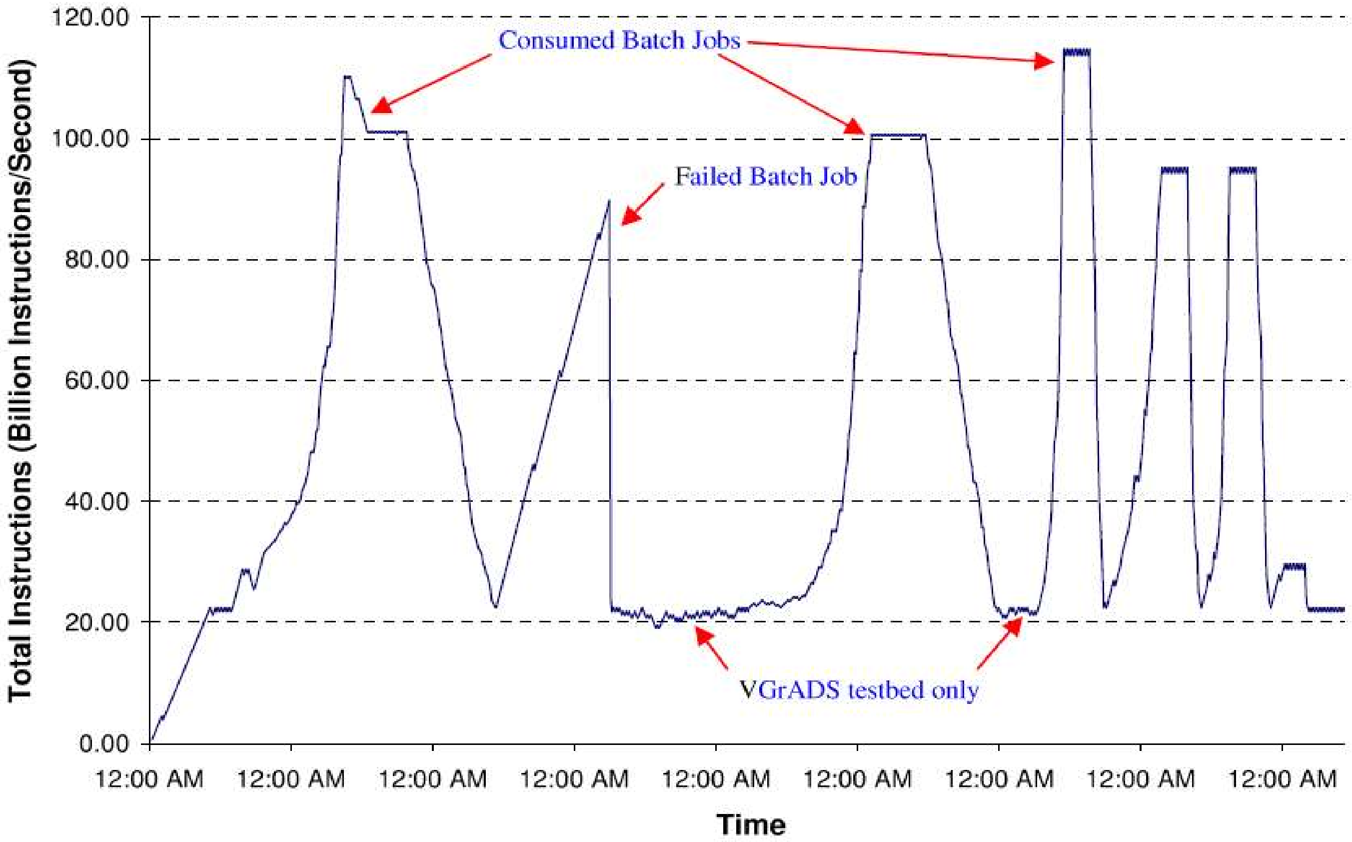
Resource usage: Hanoi6, six days



UCSB



Total IPS:



UCSB

Conclusions:

- New grid application
- Grid application using first principals
- Manages heterogeneous resources over long periods
- Use large compute power to solve previously unsolved problems

UCSB

Thanks

- LRAC Allocation through NSF
- TeraGrid:
  - SDSC, NCSA, PSC, TACC
- DataStar at SDSC: also BlueHorizon
- Mayhem Lab at UCSB