Project AutoMate

Enabling Autonomic Computational Applications on the Grid


The AutoMate Group
The Applied Software Systems Laboratory
Rutgers, The State University of New Jersey
http://automate.rutgers.edu
Ack: NSF (CAREER, KDI, ITR, NGS), DoE (ASCI, CIT)

Autonomic Computing Workshop
25 June 2003, Seattle, WA
Outline

• Autonomic Grid computing – motivations and overview
• AutoMate: A framework of enabling autonomic applications
  – ACCORD: Autonomic component framework
  – RUDDER: Decentralized deductive engine
  – SESAME: Context sensitive dynamics access management
  – Pawn: Peer-to-Peer messaging infrastructure
  – SQUID: Decentralized discovery service
• Application Scenarios
  – Autonomic interactions oil reservoir optimization
  – vGrid autonomic runtime for adaptive applications
    • reactive/proactive partitioning, load-balancing, scheduling, performance management

Conclusions
Computational Modeling of Physical Phenomenon

• Realistic, physically accurate computational modeling
  – Large computation requirements
    • e.g. simulation of the core-collapse of supernovae in 3D with reasonable resolution \((500^3)\) would require \(\sim 10-20\) teraflops for 1.5 months (i.e. \(\sim 100\) Million CPUs!) and about 200 terabytes of storage
    • e.g. turbulent flow simulations using active flow control in aerospace and biomedical engineering requires \(5000 \times 1000 \times 500 = 2.5 \times 10^9\) points and approximately 107 time steps, i.e. with 1GFlop processors requires a runtime of \(\sim 7 \times 10^6\) CPU hours, or about one month on 10,000 CPUs! (with perfect speedup). Also with 700B/pt the memory requirement is \(\sim 1.75TB\) of run time memory and \(\sim 800TB\) of storage.
  – Complex couplings
    • multi-physics, multi-model, multi-resolution, ….
  – Complex interactions
    • application – application, application – resource, application – data, application – user, …
  – Software/systems engineering/programmability
    • volume and complexity of code, community of developers, …
      – scores of models, hundreds of components, millions of lines of code, …
The Grid

• The Computational Grid
  – Potential for aggregating resources
    • computational requirements
  – Potential for seamless interactions
    • new applications formulations

• Developing application to utilize and exploit the Grid remains a significant challenge
  – The problem: a level of complexity, heterogeneity, and dynamism for which our programming environments and infrastructure are becoming unmanageable, brittle and insecure
    • System size, heterogeneity, dynamics, reliability, availability, usability
    • Currently typically proof-of-concept demos by “hero programmers”
  – Requires fundamental changes in how applications are formulated, composed and managed
    • Breaks current paradigms based on passive components and static compositions
    • autonomic components and their dynamic composition, opportunistic interactions, virtual runtime, …
  – Resonance - heterogeneity and dynamics must match and exploit the heterogeneous and dynamic nature of the Grid

• Autonomic, adaptive, interactive Grid application offer the potential solutions
  – Autonomic: context aware, self configuring, self adapting, self optimizing, self healing,…
  – Adaptive: resolution, algorithms, execution, scheduling, …
  – Interactive: peer interactions between computational objects and users, data, resources, …
AutoMate: Enabling Autonomic Applications (http://automate.rutgers.edu)

• Objective:
  – To enable the development of autonomic Grid applications that are context aware and are capable of self-configuring, self-composing, self-optimizing and self-adapting.

• Overview:
  – Definition of Autonomic Components:
    • definition of programming abstractions and supporting infrastructure that will enable the definition of autonomic components
    • autonomic components provide enhanced profiles or contracts that encapsulate their functional, operational, and control aspects
  – Dynamic Composition of Autonomic Applications:
    • mechanisms and supporting infrastructure to enable autonomic applications to be dynamically and opportunistically composed from autonomic components
    • compositions will be based on policies and constraints that are defined, deployed and executed at run time, and will be aware of available Grid resources (systems, services, storage, data) and components, and their current states, requirements, and capabilities
  – Autonomic Middleware Services:
    • design, development, and deployment of key services on top of the Grid middleware infrastructure to support autonomic applications
    • a key requirements for autonomic behavior and dynamic compositions is the ability of the components, applications and resources (systems, services, storage, data) to interact as peers
AutoMate: Architecture

• Key components:
  – **Accord**: Autonomic application framework
  – **Rudder**: Decentralized deductive engine
  – **Squid**: P2P discovery service (C. Schmidt, HPDC 2003)
  – **SESAME**: Dynamic access control engine
  – **Pawn**: P2P messaging substrate (V. Matossian, CLADE 2003)
AutoMate: Architecture

• **AutoMate System Layer:**
  – builds on the Grid middleware and OGSA and extends core Grid services to support autonomic behavior
  – provide specialized services such as peer-to-peer semantic messaging, events and notification

• **AutoMate Component Layer:**
  – addresses the definition, execution and runtime management of autonomic components
  – provides supporting services such as discovery, factory, lifecycle, context, etc.

• **AutoMate Application Layer:**
  – builds on the component and system layers to support the autonomic composition and dynamic (opportunistic) interactions between components

• **AutoMate Engines:**
  – decentralized (peer-to-peer) networks of agents in the system.
    • context-awareness engine composed of context agents and services and provides context information at different levels to trigger autonomic behaviors
    • deductive engine composed of rule agents which are part of the applications, components, services and resources, and provides the collective decision making capability to enable autonomic behavior
    • trust and access control engine composed of access control agents and provides dynamic context-aware control to all interactions in the system

• **AutoMate Portals**
  – provide users with secure, pervasive (and collaborative) access to the different entities
  – enable users to access resource, monitor, interact with, and steer components, compose and deploy applications, configure and deploy rules, etc.
ACCORD: Autonomic Components

- Autonomic components export information and policies about their behavior, resource requirements, performance, interactivity and adaptability to system and application dynamics
  - functional aspects
    - abstracts component functionality, such as order of interpolation (linear, quadratic, etc.)
    - used by the compositional engine to select appropriate components based on application requirements
  - operational aspects
    - abstracts a component's operational behavior, including computational complexity, resource requirements, and performance (scalability)
    - used by the configuration and runtime engines to optimize component selection, mapping and adaptation
  - control aspect
    - describes the adaptability of the component and defines sensors/actuators and policies for management, interaction and control.
ACCORD: Autonomic Components

- Autonomic components encapsulate access policies, rules, a rule agent, and an access agent
  - enables components to consistently and securely configure, manage, adapt and optimize their execution based on rules and access policies.
  - rules/policies can be dynamically defined (and changed) in terms of the component’s interfaces (based on access policies) and system and environmental parameters
  - rule execution may change the state, context and behavior of a component, and can generate events to trigger other rule agents
  - rule agent manages rule execution and resolves rule conflicts
ACCORD: Self-Management Approaches

• Passive:
  – Provide sensors for external accesses to collect component information
  – Provide actuators for external operations to control component behavior

• Active:
  – Collect external (local) status information through self-observation or collective-observation. Collect internal status information through sensors
  – Corresponding actions are issued based on this information in accordance with defined rules/policies/constraints

• Proactive:
  – Automatically adjust behavior in anticipation of future problems, needs or changes, based on history and/or predictive functions.
**ACCORD: Autonomic Components – Prototype implementation (EuroPar 03)**

![Diagram of Computational node, Autonomic component, RuleAgent, access, policies, agents, RA, control interface, rule interface, functional interface, Rule engine, Gateway, RA, access policies actuators rules, sensors, rule operations, Computational object]

AMS 2003, 25 June 2003
ACCORD: Autonomic Composition Engine

- Dynamically synthesize a service/composition plan at runtime based on dynamically defined goals, constraints and context
  - annotate services/components with semantic information describing its functionality and interfaces
  - use relational algebra to choreograph ad hoc interactions
  - use constraints to define and evaluate composition/service plans
Algorithm

[Initialization]
1. Each service description document is parsed for metadata
2. Semantic information is used to enhance service description
3. Composition request is made by the composer, consists of
   1. Composition Objective
   2. Semantic metadata
   3. Semantic threshold value i.e. the degree of correlation expected
   4. Constraints
   5. Start and target operations/services

[Selection]
1. Select appropriate services based on semantic matching
2. Executing constraints to refine services selection and composition
Algorithm

[Plan Generation]

1. All possible ad-hoc interactions are formulated
2. Service Graph is constructed, where
   1. Each operation acts as vertices of abstract graph
   2. The output argument types of operation are matched with the input parameters. If matching is correct, interaction link is created between operations
3. Constraints are executed to enable or disable inconsistent interactions
4. Initial and final operations are selected/specifed based on semantic information of the composition and composition graph is created
5. Composition plan(s) is(are) generated or status is returned
   1. Path in the interaction graph from source to destination operation corresponds to sequence of required message invocations.
   2. Operations lying on the path correspond to participating services.
   3. Scenarios where multiple composition plans can exist, the cost factor is evaluated for each path and least cost path is selected
ACCORD: ACE - Prototype operation

1. Composition request
   - Objective
   - Constraints
   - Semantic metadata

2. Connect and select services
   - Based on constraints
   - Based on keywords
   - Based on input arguments

3. Create interaction links
   - Using relational join based on semantic annotations

4. Synthesize composition plans as paths in the ad hoc service graph

5. Rank and return composition plans

Service Pool

ACE Agent

AMS 2003, 25 June 2003
ACE Architecture

ACE Translator → Graph Generator → Constraint Analyzer → Plan Generator

ACCORD Composition Engine

<table>
<thead>
<tr>
<th>SourceOperation</th>
<th>SourceService</th>
<th>SourceMessageName</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetOperation</td>
<td>TargetService</td>
<td>TargetMessageName</td>
</tr>
<tr>
<td>CostOfLink</td>
<td>Valid</td>
<td>Level</td>
</tr>
</tbody>
</table>

Service Table

<table>
<thead>
<tr>
<th>ServiceName</th>
<th>Objective</th>
<th>Keywords</th>
</tr>
</thead>
</table>

Node Table

<table>
<thead>
<tr>
<th>ServiceName</th>
<th>Operation</th>
<th>ParamOrder</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputMessage</td>
<td>OutputMessage</td>
<td></td>
</tr>
</tbody>
</table>

Message Table

<table>
<thead>
<tr>
<th>MessageName</th>
<th>ArgumentTypes</th>
<th>ArgumentName</th>
</tr>
</thead>
</table>
ACCORD: Opportunistic Interactions

- Interactions based on local goals and objectives
  - local goals and objectives are defined as constraints to be satisfied
  - constraints can updated and new constraints can defined at any time

- Dynamic and ad-hoc
  - interactions use “semantic messaging” based on proximity, privileges, capabilities, context, interests, offerings, etc.

- Opportunistic
  - constraints are long-term and satisfied opportunistically (may not be satisfied)

- Probabilistic guarantees and soft state
  - no explicit synchronization
  - interaction semantics are achieved using feedback and consensus building
RUDDER: The AutoMate Deductive Engine

• RUDDER is a decentralized deductive engine composed of distributed specialized agents (component rule agents, composition agents, context agents and system agents) that exist at different levels of the system, and represents their collective behavior.

• Objectives
  – Providing mechanisms for dynamically defining, configuring, deploying rules, and rule conflicts management
  – Runtime management services, supporting autonomic composition, adaptation, optimization and execution
RUDDER: Agent Architecture

- Goal-directed focus: focus on the objective and choose the method to achieve it
- Context sensitivity: make decisions about what to try and retry based on present conditions
BDI Agent Model

- An agent has beliefs about the world and desires to satisfy, driving it to form intentions to act
  - Beliefs: about the environment and other agents
  - Desire or goals to achieve
  - Intention or plans to act upon to achieve its desires

Human ➔ Belief, Desire, Intentions Agent

- Beliefs - perceived understanding of the world
- Goals or desires
- Accumulated experience and behaviours

- Beliefs - database of perceived world knowledge
- Goals or desires
- Intentions - currently executing plans

- Pre-compiled plans

AMS 2003, 25 June 2003
Agent Hierarchy Construction

Goal

Subgoal1
Component/Services
Component/Services
Component/Services
Component/Services
Component/Services
Component/Services

Subgoal2
Component/Services
Component/Services
Component/Services
Component/Services
Component/Services
Component/Services

Subgoal3
Component/Services
Component/Services
Component/Services
Component/Services
Component/Services
Component/Services

Composer Agent
Composer Agent
Composer Agent
Composer Agent
Composer Agent
Composer Agent

Application Agent
Component Agent
Component Agent
Component Agent
Component Agent
Component Agent

AMS 2003, 25 June 2003
SESAME: Context Aware Access Management (CCS 03)

- **Objective:**
  - support dynamic, seamless and secure interactions between the participating entities (i.e. components, services, application, data, instruments, resources and users)
  - Autonomic Computing – Self Protecting (Context aware, Dynamic)

- **Issues:**
  - access rights in highly dynamic and heterogeneous Grid environments depends on the entity's privileges, capabilities, context and state
    - e.g. the ability of a user to access a resource or steer a component depends on users' privileges (e.g. owner), current capabilities (e.g. resources available), current context (e.g. location, time, secure connection) and the state of the resource or component

- **Approach**
  - extend Role Based Access Control (RBAS) to make access control decision based on dynamic context information
  - dynamically adjust Role Assignments and Permission Assignments based on context
SESAME: Operation

- Dynamically adjusts the user-role and role-permission relationships based on context information
  - each component is assigned a role subset (by the authority service) from the entire role set on authentication
  - each component maintains permission subsets for each role that will access the component
  - during an interaction, state machines are maintained by the delegated access control agent at the subject (Role State Machine) to navigate the role subset, and the object (Permission State Machine) to navigate the permission subset for each active role
  - state machines define the currently active role permissions
  - access agent navigates the role/permission subsets to react to changes in the context
The access control agent maintains the role state machine for each component and defines its active role based on its current context.

When the subject component accesses another component, it will first get its current role from its role state machine, and then use this role to access the component.

At the accessed component, a permission state machine is defined (if it does not already exist) for the active role.

For example, active roles X, Y, and Z have their own permission state machines at component. The access control agent at the accessed component will maintain this permission state machine to define the current permissions for a role based in its current context and state.
Role & Permission State Machine

Role Hierarchy

Permission Hierarchy
SQUID: A Decentralized Discovery Service

• Overview/Motivation:
  – Efficient information discovery in the absence of global knowledge of naming conventions is a fundamental problem in large, decentralized, distributed resource sharing environments such as the Grid.
    • a document is better described by keywords than by its filename, a computer by a set of attributes such as CPU type, memory, operating system type than by its host name, and a component by its aspects than by its instance name.
  – Heterogeneous nature and large volume of data and resources, their dynamism (e.g. CPU load) and the dynamism of the Grid make the information discovery a challenging problem.

• Key features
  – P2P system that supports complex queries containing partial keywords, wildcards, and range queries.
  – Guarantees that all existing data elements that match a query will be found with bounded costs in terms of number of messages and number of nodes involved.
  – The system can be used as a complement for current resource discovery mechanisms in Computational Grids (to enhance them with range queries).
SQUID: Design

- Overall architecture is a distributed hash table (DHT), similar to typical data lookup systems (e.g. Chord, CAN)
- Key innovation is a locality preserving, dimension reducing indexing scheme that effectively maps the multidimensional information space to physical peers
  - data elements described using a sequence of keywords (common words in the case of P2P storage systems, or values of globally defined attributes - such as memory and CPU frequency - for resource discovery in computational grids)
    - keywords form a multidimensional keyword space where the keywords are the coordinates and the data elements are points in the space.
    - two data elements are “local” if their keywords are lexicographically close or they have common keywords
  - use Space Filling Curves to map documents that are local multi-dimensional index space to indices that are local in the 1-dimensional index space
    - load-balancing at join and runtime
  - existing systems, this is done using consistent hashing to uniformly map data element identifiers to indices
    - data elements are randomly distributed across peers without any notion of locality
SQUID: Operation
Pawn: A P2P Messaging Substrate

- **Objective**
  - Engineer a peer-to-peer messaging substrate that extends existing solutions to enable high-level interactions for scientific applications.

- **Architecture**
  - Peers, Messages, Services, Interactions

- **Key Features**
  - Stateful messages
  - Guaranteed messaging semantics
  - Publish/subscribe mechanisms across peer-to-peer domains
  - High-level messaging semantics
    - Sync/Async Messaging
    - PUSH (dynamic injection)
    - PawnRPC

- **Built on Project JXTA**
  - Pipes
  - Resolver

Interactions
- Synchronous/Asynchronous; Dynamic Data Injection; Remote Procedure Calls

Services
- Application Execution; Application Runtime Control; Application Monitoring and Steering; Collaboration

Messages
- Platform-independent; Coordination; Guarantees

Peers
- Client; Rendezvous; Application
Autonomic Oil Well Placement (UT-CSM, UT-IG)

- Optimization algorithm: use VFSA (Very Fast Simulated Annealing)
  - requires function evaluation only, no gradients
- IPARS delivers
  - fast-forward model (guess->objective function value)
  - post-processing
- Formulate a parameter space
  - well position and pressure \((y,z,P)\)
- Formulate an objective function:
  - maximize economic value \(\text{Eval}(y,z,P)(T)\)
- Normalize the objective function \(\text{NEval}(y,z,P)\) so that:

\[
\min \text{Neval}(y,z,P) \iff \max \text{Eval}(y,z,P)
\]
Components of the AORO Application

- **IPARS**: Integrated Parallel Accurate Reservoir Simulator
  - Parallel reservoir simulation framework
- **IPARS Factory**
  - Configures instances of IPARS simulations
  - Deploys them on resources on the Grid
  - Manages their execution
- **VFSA**: Very Fast Simulated Annealing
  - Optimizes the placement of wells and the inputs (pressure, temperature) to IPARS simulations.
- **Economic Modeling Service**
  - Uses IPARS simulations outputs and current market parameters (oil prices, costs, etc.) to compute estimated revenues for a particular reservoir configuration.
- **DISCOVER Computational Collaboratory**
  - Interaction & Collaboration
  - Distributed Interactive Object Substrate (DIOS)
  - Collaborative Portals
The diagram illustrates the interaction between Clients, IPARS Factory, VFSA Optimization Service, and PAWN Substrate. Here is the described process:

2. IPARS Factory discovers and initializes VFSA Optimization Service.
3. Client can configure IPARS parameters.
4. IPARS Factory gets initial guess from VFSA Optimization Service and launches IPARS instance on resource of choice.
5. IPARS connects to VFSA Optimization Service and presents revenue.
6. VFSA Optimization Service generates new well placement.
7. One optimal well placement is determined, IPARS Factory launches IPARS run.
8. Scientists/Engineers collaboratively interact with IPARS.

Current oil price, market state, etc.
Autonomic Oil Well Placement

Permeability

Pressure contours
3 wells, 2D profile

Contours of $\text{NEval}(y,z,500)(10)$

Requires $NY \times NZ$ (450) evaluations. Minimum appears here.

VFSA solution: “walk”:
found after 20 (81) evaluations
Sample Results

Guess Y 264.57623  Guess Z 170.17809

Y/Z positions for Y&Z [0,315]

First guess on injection well position

(Final) Iterative guesses optimizing revenue

Fixed Injection well

Production wells

Minimizing cost

Accepted Guesses

AMS 2003, 25 June 2003
ARMaDA: Adaptive Partitioning and Optimization for SAMR Applications

- Partitioning, load-balancing and scheduling of SAMR applications.
  - **Partitioning Scheme**
    - “Best” partitioning based on application/system configuration and current application/system state
      - G-MISP+SP, pBD-ISP, SFC (Vampire, GrACE, Zoltan, ParMetis, …)
  - **Granularity**
    - patch size, AMR efficiency, comm./comp. ratio, overhead, node-performance, load-balance, …
  - **Number of processors/Load per processor**
    - Dynamic allocations/configuration/management
  - Hierarchical decomposition using dynamics processor groups
  - Communication optimizations/latency tolerance/multithreading
  - Availability, capabilities, and state of system resources
A Selection of SAMR Application Enabled

Blast wave in the presence of a uniform magnetic field) – 3 levels of refinement. (Zeus + GrACE + Cactus, P. Li, NCSA, UCSD)

Multi-block grid structure and oil concentrations contours (IPARS, M. Peszynska, UT Austin)

Mixture of H2 and Air in stoichiometric proportions with a non-uniform temperature field (GrACE + CCA, Jaideep Ray, SNL, Livermore)

Richtmyer-Meshkov - detonation in a deforming tube - 3 levels. Z=0 plane visualized on the right (VTF + GrACE, R. Samtaney, CIT)
ARMaDA: Application-sensitive Adaptations

- PAC tuple, 5-component metric
- Octant approach: app. runtime state
- GrACE (ISP), Vampire (pBD-ISP, GMISP+SP) partitioners
- ARMaDA framework
  - Computation/communication
  - Application dynamics
  - Nature of adaptation
- RM3D, 64 procs on “Blue Horizon”
  - 100 steps, base grid 128*32*32
  - 3 levels, RF = 2, regrid 4 steps

ARMaDA evaluation for VectorWave-2D application on 32 processors on “Frea”

<table>
<thead>
<tr>
<th>Partitioner</th>
<th>Execution time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFC</td>
<td>637.478</td>
</tr>
<tr>
<td>G-MISP+SP</td>
<td>611.749</td>
</tr>
<tr>
<td>pBD-ISP</td>
<td>592.05</td>
</tr>
<tr>
<td>ARMaDA with SFC start</td>
<td>470.531</td>
</tr>
</tbody>
</table>
ARMaDA: System-sensitive Adaptations

- System characteristics using NWS
- RM3D compressible turbulence application
  - 128x64x64 base (coarse) grid
  - 3 levels, factor 2 refinement
- System/Environment
  - University of Texas at Austin (32 nodes), Rutgers (16 nodes)

\[
C_k = w_p P_k + w_m M_k + w_b B_k
\]

<table>
<thead>
<tr>
<th>Procs</th>
<th>Dynamic Sensing (s)</th>
<th>Static Sensing (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>423.7</td>
<td>805.5</td>
</tr>
<tr>
<td>4</td>
<td>292</td>
<td>450</td>
</tr>
<tr>
<td>6</td>
<td>272</td>
<td>424</td>
</tr>
<tr>
<td>8</td>
<td>225</td>
<td>430</td>
</tr>
</tbody>
</table>

Execution time (secs)

Number of processors
Autonomic Computational Science and Engineering

Autonomic Computing
- Simulation of Active Flow Control of Turbulent Flows
- Virtual Groundwater Basin Model

vGrid Application Development
- Thermonuclear Combustion Supernovae
- Non-Born - Oppenheimer Molecular Quantum Mechanics

Analyse
Plan
Monitor
Knowledge
Execute

vGrid Runtime Management

vGrid Grid Virtualization Services
- Temporal Scheduling
- Spatial Scheduling

Grid Middleware
Autonomic Runtime Management: “Working Sets”

High computation zone
Application Runtime Management in V-Grid

Loop for each level of Grid/Application hierarchy

V-Grid Monitoring
(Self-observation, Context-awareness)
- System states (CPU, Memory, Bandwidth, Availability etc.)
- Application states
  (Computation/Communication Ratio, Nature of Applications, Application Dynamics)

V-Grid Deduction
(Self-adaptation, Self-optimization, Self-healing)
- Identify and characterize natural regions
- Define objective functions and management strategy
- Define VCU

V-Grid Execution
Partition, Map and Tune

Virtual Grid Resource Autonomic Runtime Manager (ARM)

Grid Resource Hierarchy

Application Domain Hierarchy

NR: Application Natural Regions
VCU: Virtual Computational Unit

Self-learning
Autonomic Runtime Management

- Dynamic Driver Application
- Self-Observation & Analysis
  - Application State Characterization
  - Natural Region Characterization
  - Objective Function Synthesizer
  - Prescriptions
  - Normalized Work Metric
  - NRM
  - NWM
  - Autonomic Partitioning
    - Virtual Grid Space Scheduling (VGSS)
    - Virtual Grid Time Scheduling (VGTS)
  - Mapping
  - Global Grid Scheduling
  - Local Grid Scheduling
  - Autonomic Scheduling
  - Virtual Computation Unit

- Application Monitoring Service
- Resource Monitoring Service
- Monitoring & Context-Aware Services
  - Resource History Module
  - System Synthesizer
  - System Capability Module
  - Performance Prediction Module
  - Resource History Module
  - System State Synthesizer
  - System Capability Module

- Nature of Adaptation
- Application Dynamics

- CPU
- Memory
- Bandwidth
- Availability
- Access Policy

- Heterogeneous, Dynamic Computational Environment

- Decision Space
  - Communication Overheads
  - Adaptive Partitioning
  - Memory Requirement
  - Load Balancing
  - Granularity Control

- Autonomic Scheduling
  - Global Grid Scheduling
  - Local Grid Scheduling
  - VGTS: Virtual Grid Time Scheduling
  - VGSS: Virtual Grid Space Scheduling
Conclusion

• Autonomic applications necessary to address scale/complexity/heterogeneity/dynamism/reliability challenges

• AutoMate addresses key issues to enable the development of autonomic Grid applications
  – ACCORD: Autonomic application framework
  – RUDDER: Decentralized deductive engine
  – SESAME: Dynamic access control engine
  – Pawn: P2P messaging substrate
  – SQUID: P2P discovery service

• Application scenarios
  – vGrid autonomic runtime management of SAMR applications
  – Autonomic optimization of oil reservoirs

• More Information, publications, software
  – http://automate.rutgers.edu
  – automate@caip.rutgers.edu / parashar@caip.rutgers.edu
The Team

- **TASSL Rutgers University**
  - Autonomic Computing Research Group
    - Viraj Bhat
    - Manish Agarwal
    - Hua Liu (Maria)
    - Zhen Li (Jenny)
    - Manish Mahajan
    - Vincent Matossian
    - Venkatesh Putty
    - Cristina Schmidt
    - Guangsen Zhang
  - Autonomic Applications Research Group
    - Sumir Chandra
    - Xiaolin Li
    - Taher Saif
    - Li Zhang
    - Hailan Zhu

- **CS Collaborators**
  - HPDC, University of Arizona
    - Salim Hariri
  - Biomedical Informatics, The Ohio State University
    - Tahsin Kurc, Joel Saltz
  - CS, University of Maryland
    - Alan Sussman, Christian Hansen

- **Applications Collaborators**
  - CSM, University of Texas at Austin
    - Malgorzata Peszynska, Mary Wheeler
  - IG, University of Texas at Austin
    - Mrinal Sen, Paul Stoffa
  - ASCI/CACR, Caltech
    - Michael Aivazis, Julian Cummings, Dan Meiron
  - CRL, Sandia National Laboratory, Livermore
    - Jaideep Ray, Johan Steensland