Investigating Autonomic Behaviours in Grid-Based Computational Science Applications

Shantenu Jha
Center for Computation & Technology, Louisiana State University, USA
Department of Computer Science, Louisiana State University, USA
e-Science Institute, University of Edinburgh, UK
sjha@cct.lsu.edu

Manish Parashar
NSF Center for Autonomic Computing
Department of Electrical and Computer Engineering
Rutgers University, USA
parashar@rutgers.edu

Omer Rana
School of Computer Science
Cardiff University, UK
o.f.rana@cs.cardiff.ac.uk

ABSTRACT

Emerging Grid infrastructures present unprecedented opportunities for computational science and engineering, with the potential for fundamental insights into complex phenomenon. However, it also presents unprecedented challenges in terms of its scale, heterogeneity, dynamism and overall complexity that must be addressed before this potential can be realized. Autonomic computing concepts have been effectively used to address similar challenges in enterprise systems and applications; in this paper, we explore the role of autonomic computing to Grid-based computational science applications. Specifically, we use three representative computational applications to motivate Autonomic Computational Science (ACS). Using these applications, we develop a conceptual framework for ACS, consisting of *mechanisms, strategies and objectives*, and demonstrate how these concepts can be used to express autonomic behaviors. Finally we explore a research agenda towards realizing ACS behaviors and developing self-* patterns for large scale autonomic computational science application.

Categories and Subject Descriptors

D [**Software**]: Miscellaneous

Keywords

Computational Science, Applications, Autonomics

1. INTRODUCTION

Significant investment in national and global cyberinfrastructure, can enable seamless, secure, on-demand access to, and aggregation of, geographically distributed computing, communication and information resources. Such advances have the potential for enabling important and heretofore unimaginable scientific progress and insight. However the ability of scientists to realize this potential is being severely hampered, primarily due to the increasing complexity and dynamism of the applications and the underlying computing environments. In fact, to be productive, scientists often have to comprehend and manage complex computing configurations, software tools and libraries as well as application parameters and behaviors. As a result, the number of applications that actually use Grid deployments such as the US TeraGrid and Open Science Grid (OSG), the UK National Grid Service (NGS), or the European DEISA and EGEE, in novel ways is limited.

Autonomic computing concepts [1], which have been effectively used to manage and optimize enterprise systems and applications, provide a promising approach that can be used to address the challenges outlined above. Autonomic computing is inspired by biological systems, and aims at developing systems and application that can manage and optimize themselves using only high-level guidance or interference from users. Autonomic computing systems dynamically adapt to changes in accordance with business policies and objectives and take care of routine elements of management similar to the unconscious self-regulation behavior of biological systems.

The overarching objective of this paper is to investigate how computational science applications can benefit form autonomic computing concepts, while at the same time exploring how computational science can present new application opportunities and challenges to autonomic computing. To achieve this objective we take an application-centric approach. Specifically, we focus on three applications, which represent broader application classes, and highlight how autonomics can be used to address specific requirements and challenges in these applications, demonstrate potential benefits for using autonomics, and highlight gaps that must be filled before autonomic solutions can be broadly deployed for these application classes. Note that autonomics can also be used to manage Grid infrastructure that supports computational science (analogous to enterprise infrastructures). The focus of this paper however, is on the application and autonomics driven from the application perspective.

This paper is motivated by our recent paper – Abstractions for Large-Scale Distributed Applications and Systems (Ref [2]) – where we studied the structure of "real" Grid applications and identified recurring patterns in these applications, as well as their underlying requirements and challenges. This paper builds on the conceptual understanding gained in Ref [2], and investigates how autonomics can be used to address these challenges – the longer term goal of this research is to investigate similar recurring and crosscutting autonomic computing patterns with respect to computational science applications, i.e., "self-* patterns".

We note that application and system specific runtime adaptations are widely used in computational science applications as a means for managing applications and tuning performance. Similarly, existing tools can automatically generate parameters, for example, for classes for numerical applications to optimize them for specific processors. Our goal in exploring autonomic computation science is not to reinvent such strategies, but rather to enable developers to reason about and design such adaptations in a more structured and flexible manner instead of an ad hoc afterthought. The key underlying concept is the separation of management and optimization policies from enabling mechanisms that allows a repertoire of a mechanisms to be automatically orchestrated at runtime to respond to the heterogeneity and dynamics, both of the applications and the infrastructure. Examples of mechanisms could be alternative numerical algorithms, domain decompositions, and communication protocols; an example of a policy is to select a latency-tolerant algorithm when network load is above certain thresholds. Using this concept, we will develop strategies that are capable of identifying and characterizing patterns at design and at runtime and, using relevant policies, managing and optimizing the patterns. For example, dynamic, proactive management services can use such a strategy to optimize resource utilization and application performance in situations where computational characteristics and/or resource characteristics may change. For instance, if using adaptive mesh refinement increases computational costs, the autonomic framework may negotiate to obtain additional resources or to reduce resolution, depending on resource availability and user preferences.

The rest of this paper is structured as follows. Section 2 presents a conceptual framework for thinking about autonomic computing in the context of computational science applications and defines key concepts and terms. Section 3 discuss three representative applications and investigates the role of autonomics and demonstrates the conceptual framework in the context of these applications. This section concludes with a discussion about crosscutting autonomic behaviors and potential patterns. Section 4 concludes the paper by highlighting key challenges as well as potential opportunities.

Note that the contribution of this paper lies in our application-driven investigation of the role of autonomics in realizing the potential of Grids, which is driven by a detailed analysis of real-world scientific applications. Through this analysis we develop a conceptual framework that can be used to reason about autonomic behaviors across all aspects of the application including development, deployment and execution.

2. A CONCEPTUAL FRAMEWORK FOR AUTONOMIC COMPUTATIONAL SCIENCE

2.1 The Autonomic Computing Paradigm

The autonomic computing paradigm is modeled after the autonomic nervous system, and enables changes in its essential variables (e.g., performance, fault, security, etc.) to trigger changes to the behavior of the computing system such that the system is brought back into equilibrium with respect to the environment [3]. Conceptually, an autonomic system requires: (a) sensor channels to sense the changes in the internal and external environments, and (b) motor channels to react to and counter the effects of the changes in the environment by changing the system and maintaining equilibrium. The changes sensed by the sensor channels have to be analyzed to determine if any of the essential variables have gone out of their viability limits. If so, it has to trigger some kind of planning to determine what changes to inject into the current behavior of the system such that it returns to the equilibrium state within the new environment. This planning requires knowledge to select the right behavior from a large set of possible behaviors to counter the change. Finally, the motor neurons execute the selected change. Sensing, Analyzing, Planning, Knowledge and Execution are thus the keywords used to identify an autonomic computing system and define the *MAPE* (Monitor-Analyze-Plan-Execute) [4] as well as other models for autonomic computing [1]. In what follows, we explore the applications of this paradigm to support computational science on Grid infrastructure, as well as to explore manifestations of autonomic architectures and behaviors in Grid-based ACS applications.

2.2 Conceptual Architectures for ACS

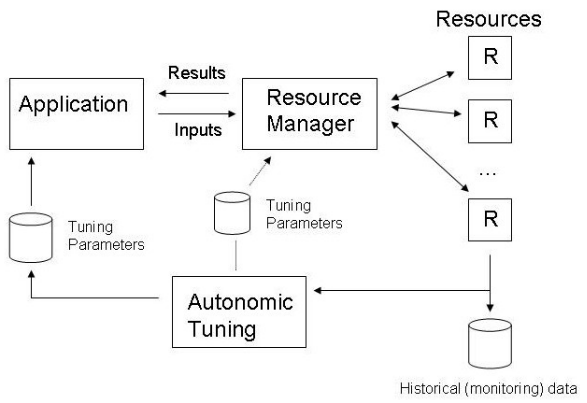

Figure 1: Autonomic tuning *of* application & resource manager parameters

Looking at existing practices in computational science, two corresponding conceptual architectures can be observed, which are described below. These architectures are composed of the application, a resource manager that allocates, configures and tunes resources for the application, and an autonomic manager that performs the autonomic tuning of application and/or system parameters. Figure 1 illustrates the first conceptual architecture, where the application and resources are characterized using a number of *dynamically modifiable* parameters/variables that have an impact on the overall *observed behaviour* of the application. Each of these parameters has an associated range over which it can be modified, and these constraints are known *a priori*. The intention of the autonomic tuning engine is to alter these parameters based on some overall *required behaviour* (hereby referred to as the application objective) that has been defined by the user. Tuning in this case is achieved by taking into account, for example, (i) historical data about previous runs of the application on known resources, obtained using monitoring probes on resources; (ii) historical data about previous selected values of the tunable parameters; (iii) empirically derived models of application behavior; (iv) the specified tuning mechanism and strategy; etc.

For example, an autonomic tuning mechanism in this architecture may involve changing the *size* of the application (for instance, the number of data partitions generated from a large data set, the number of tiles from an image, etc.), or the set of parameters over which execution is being requested. This tuning is used to make desired tradeoffs between quality of solution, resource requirements and execution time or to ensure that a particular Quality of Service (QoS) constraint, such as execution time, is satisfied.

A variant, also illustrated in Figure 1, involves updating the resource manager based on information about the current state of the application. As an example of such an approach, consider an application using dynamic structured adaptive mesh refinement (SAMR) [5] techniques on structured meshes/grids. Compared to numerical techniques based on static uniform discretization, SAMR methods employ locally optimal approximations and can yield highly advantageous ratios for cost/accuracy by adaptively concentrating computational effort and resources to regions with large local solution error at runtime. The adaptive nature and inherent space-time heterogeneity of these SAMR implementations lead to significant challenges in dynamic resource allocation, data-distribution, load balancing, and runtime management. Identifying how the underlying resource management infrastructure should adapt to changing SAMR requirements (and possibly vice versa) provides one example of the architecture in Figure 1.

Figure 2 illustrates another conceptual architecture, where the application is responsible for driving the tuning of parameters, and choosing a tuning strategy. The autonomic manager is now responsible for obtaining monitoring data from resource probes and the strategy specification (for one or more objectives to be realized) from the application. Tuning now involves choosing a resource management strategy that can satisfy the objectives identified by the application. This approach primarily relates to the *system-level* self-management described in Section 1. An example of such an architectural approach is the use of resource reservation to achieve a particular QoS requirement. The G-QoSM framework [6] demonstrates the use of such an architecture, involving the use of a soft real-time scheduler (DSRT) along with a bandwidth broker to make resource reservation over local compute, disk and network capacity, in order to achieve particular application QoS constraints.

2.3 The ACS Conceptual Framework

A conceptual framework for ACS can be developed based on the conceptual architectures discussed above (and illustrated in Figures 1 and 2), comprised of the following elements:

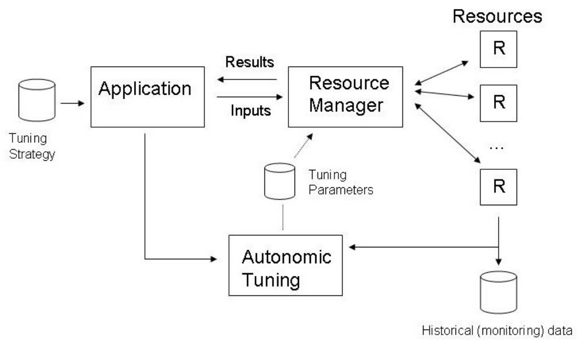

Figure 2: Autonomic tuning *by* application

- Application-level Objective (AO): An AO refers to an application requirement that has been identified by a user. Examples of AO include increase throughput, reduce task failure, balance load, etc. An AO needs to be communicated to the autonomic tuning component illustrated in Figures 1 and 2. The autonomic tuning component must then interact with a resource manager to achieve these objectives (where possible).

- Mechanism: a mechanism refers to a particular action that can be used by the application or the resource manager (from Figures 1 and 2) to achieve an AO. A mechanism m may be characterized in terms of: $\{m_i^e\}$ – the set of events that lead to the triggering (activation) of the mechanism; $\{m_i\}$ – the set of data inputs to the mechanism; $\{m_o\}$ – the set of output data generated; and $\{m_o^e\}$ – the set of output events that are generated during the execution of the mechanism or after its completion. An example of a mechanism includes *file staging*. In this mechanism, $\{m_i\}$ corresponds to one or more file and resource references prior to the staging process has started, $\{m_o\}$ corresponds to the file references after staging has completed, $\{m_i^e\}$ refers to the input events that trigger the file staging to begin, and $\{m_o^e\}$ corresponds to output events that are generated once file staging is completed.

- Strategy: One or more strategies may be used to accomplish a particular AO. A strategy is defined in terms of one or more mechanisms, that must be executed in a particular order. A strategy may be specified manually (by a systems administrator, for instance) or constructed using an autonomic approach. The focus of this particular work is on the latter. A strategy is managed by the autonomic tuning component illustrated in Figures 1 and 2, and may be maintained as a collection of templates that are adapted depending on the application or the resource manager properties.

Note that given AO can be fulfilled by multiple strategies, and the same strategy can be used for different AO. For example, self-configuration can be used for both load-balancing as well as higher-throughput. Figure 3 illustrates the relationship between these concepts. Table 1 observe the following mappings between (self-*) strategies and autonomic (application-level) objectives.

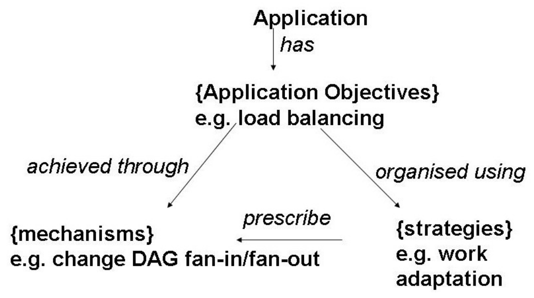

Figure 3: Relationship between AO, mechanism and strategy. An Application can have multiple application objectives (indicated by {...} notation). Each objective can be achieved through a set of mechanisms based on strategies. The set and organization of mechanisms used to achieve an AO is prescribed by the specific strategy used.

Strategies	Autonomic Objectives
Self-healing	Fault-tolerance, task-failure
Self-monitoring	Data quality
Self-configuration	Deployment, load balancing, throughput, data quality, choosing parameters
Self-optimization	Load balancing, throughput, improving performance
Self-protection	Data-access level, communication security-level

Table 1: Linking (Self-*) Strategies with Autonomic Objectives.

2.4 Towards ACS Applications

Given the conceptual framework defined above, the next step is to identify the autonomic architecture and behaviors that are meaningful to an application, and to identify appropriate objectives, mechanisms and strategies. For example, in the previous subsection two different approaches were discussed: the autonomic tuning *of* applications and the autonomic tuning *by* applications. An obvious question that follows is: Which of these approaches is suitable for a given application? What are the relative merits of the two scenarios and their respective applicability? The following questions provide a structure for thinking about ACS applications and their possible realizations.

1. Given that an application objective can be met using different strategies, which strategies are best for a given application objective? What role do the application characteristics play in determining the strategies? Similarly, what role do application characteristics play in determining the mechanisms?

2. Which mechanisms can be used to implement autonomic behaviour at the application-level? Where in the application lifecycle do these mechanism play a role?

3. What are the underlying support and implementation tools that used to support autonomic behaviour? How can these tools and support mechanisms be shared across different applications?

In what follows, we use the formalism of Application Vectors (defined below) to help structure the answer to the above questions. Specifically we will find that application vectors are a powerful way to structure and understand the importance of application characteristics, and to identify appropriate strategies and mechanisms that and be used to achieve autonomic behaviors.

Application Vectors: An analysis of the characteristics of the distributed applications suggests that there are important issues in both the development and deployment that should be considered – such factors may be used to define a set of *application vectors* [2] (AV). Every distributed application must, at a minimum, have mechanisms to address requirements for communication, coordination and execution, which form the vectors we use: Execution Unit, Communication (Data Exchange), Coordination, and Execution Environment. These aspects comprise the AVs of an application, and the specific mechanism that a distributed application uses to provide these functionalities are the values of the AV. AV is a structured way to understand the distributed characteristics of a distributed application. Understanding the values of AV for a given distributed application helps understand the "basic distributed structure" of the application.

The first vector is *execution unit*, and refers to the set of pieces/executable units of the application that are distributed. The next vector is *communication (data exchange)*. This defines the data flow between the executions units. Data can be exchanged by messages (point-to-point, all-to-all, one-to-all, all-to-one, or group-to-group), files (possibly written by one execution unit and then read by another), stream (potentially unicast or multicast), publish/subscribe, data reduction (a subset of messaging), or through shared data. The third vector is *coordination*, which describes how the interaction between execution units is managed. Possible values for this vector include dataflow, control flow, SPMD (where the control flow in implicit in all copies of the single program), master-worker (the work done by the workers is controlled by the master), or events (runtime events cause different execution units to become active.) In the case of data flow and control flow, the overall dependencies may be described as a DAG or a forest. The last vector is *execution environment*, which captures what is needed for the application to actually execute. It often includes the requirements for instantiating and starting the execution units (which is also referred to as *deployment*) and may include the requirements for transferring data between execution units as well as other runtime issues. Examples of values of this vector include: dynamic process/task creation, workflow execution, file transfer, messaging (MPI), co-scheduling, data streaming, asynchronous data I/O, decoupled coordination support, dynamic resource and service discovery, decoupled coordination and messaging, decoupled data sharing, preemption, etc.

Table 2 describes how these vectors may be used to characterize the applications discussed here. For applications that share common values across these vectors, it is likely that the same autonomic mechanisms and/or strategy may be suitable to achieve an application objective. Specifically, in the context of ACS and this paper, we would like to understand the relationship between the values of applications vectors and specific mechanisms and strategies used, and in turn, understand which characteristics of distributed appli-

Application Example	**Execution Unit**	**Communication (Data Exchange)**	**Coordination**	**Execution Environment (for communication and execution units)**
Montage	Multiple sequential and parallel executables	Files	Dataflow (DAG)	Dynamic process creation, workflow execution, file transfer
Home-based patient monitoring and data analysis	Multiple sequential and parallel executables	Pub/sub and messaging	Data- and control-flow (DAG)	Web services and workflow
NEKTAR	Multiple, concurrent instances of single executable	Messages	SPMD	MPI, co-scheduling

Table 2: Application characteristics. From Reference [1]

cation influence the choice of autonomic mechanisms used.

There are multiple ways and levels at which Grid-based computational science applications can be made autonomic, but whether this is done entirely at the application level or at the system-level, or in a hybrid manner, remains an open question. The development of a conceptual framework, coupled with an understanding of what mechanism are available and how applications can utilize the mechanisms to realize autonomic behaviors, will help provide insight into the questions listed above. Furthermore, the analysis of multiple applications using a common conceptual framework is the first step in understanding crosscutting *patterns* that support autonomic behaviour (which we refer to as self-* patterns) and for developing abstractions that can support these self-* patterns.

Before we present a description of the specific applications we have chosen, it is worth highlighting that the range of applications we investigate span a broad range of characteristics. For example, Montage is a simple yet distributed application utilizing requiring high-throughput computing; the home-based patient monitoring application requires data-streaming and finally, Nektar and the fusion simulation are examples of a tightly-coupled yet distributed application.

3. ACS APPLICATION CASE STUDIES

In discussing the following applications, it is important to remember that these are traditional distributed applications, and may not all have autonomic behaviour at this stage. We analyze both a forward looking scenario and retrospectively identify where autonomic behaviour could provide benefit.

3.1 Case Study: Montage

3.1.1 Montage

Montage [7] is an image processing application that is built for a variety of systems, including single computers, parallel computers, and Computational Grids. Here, the Grid version is discussed. Montage takes as input a set of images taken by one or more telescopes and builds a mosaic that approximates the single image that would be taken by a single large telescope. The work that is done by Montage is encapsulated in a set of executables (normally about 10), most of which are run multiple times with different data sets. Each executable uses files for input and output. The dependencies between the executable runs are stored as a DAG, and this DAG is run using Pegasus [8] and DAGMan [9]. If the systems used to run the executables do not share a file system, various files have to be moved from the system on which they were generated to the system on which they are needed; this is handled by Pegasus. Montage does not have any real gain from running on a distributed system rather than on a cluster, but it uses distributed systems to scale processing capability above what is possible on a local cluster. The main challenge associated with doing this is mapping the executables to specific resources in a way that optimizes time-to-solution for building each mosaic and also makes best use of the computational and network resources, since using resources on multiple machines leads to a need for data transfers between processes that doesn't need to be done if the executables are run on a single machine. At the Montage home site (JPL/Caltech's IPAC), a local cluster is available for typical small Montage jobs, while large jobs are farmed out to distributed resources.

Vectors	**Mechanisms**
Coordination	Adapt DAG structure, Change Fan In/Out, Cluster Nodes, Change Task Granularity
Communication	File staging, File aggregation, File splitting, File indexing
Execution Environment	DAG execution (Mapping/Scheduling), Resource Selection/Management, Task re-execution, Task migration Storage management, File caching, File distribution, (multicast, broadcast), File re-transmission, Checkpoint/restart

Table 3: ***Tuning Mechanisms in Montage***

Table 3 describes the tuning mechanisms that could be used in the Montage application, and have been classified along the three main vectors of Coordination, Communication and Execution Environment. Essentially, these mechanisms represent actions that could be undertaken within an autonomic manager to support particular application objectives, such as load balancing, managing task failure and improving throughput (in table 4). For instance, to support the "Load Balancing" objective, the autonomic manager could: (i) adapt task mapping granularity; (ii) change fan in/out of the DAG. Adapting task granularity would imply generating a greater number of tasks from the same workflow graph (the number of nodes on which these tasks are executed does not change), therefore requiring re-scheduling of existing tasks, migration of tasks to these additional nodes, and the associated file staging. It is useful to note that each objective has multiple possible strategies by which it can be accomplished. The objectives may also have dependencies, whereby fulfilment of "Load Balancing" may adversely affect "Handling Task Failure", for instance.

3.2 Case Study: Home Based Patient Monitoring (HbPM)

Support for *individual-centered* healthcare is an important trend in modern medicine, primarily aiming to identify how particular treatments and drugs impact individuals rather than a clinical group. This is also a recognition that a "one size fits all" healthcare solution may not adequately address and differentiate between individual-specific needs and particular priorities identified in a national health plan. In this context, the timely identification of individuals who pose a high risk of developing a particular disease has become an important priority. Diabetes is one such disease, the occurrence of which in adults worldwide is estimated to rise to 5.4% by 2025, with the number of adults with diabetes in the world rising from 135 million in 1995 to 300 million in 2025. The ability to capture data directly from an individual, and subsequently analyze this for trends that could indicate potential risk to the individual, provides a significant advance over current approaches to diabetes management.

The distributed infrastructure challenge in supporting such individualized analysis stems from the: (i) increasing availability of low cost mobile devices and their interconnection to each other and the existing wired platforms; (ii) large quantities of data (from sensors and existing patient registration) that needs to be stored and analyzed; (iii) the need to account for the reliability of predictions for computing individualized risk. Such monitoring and analysis also needs to be undertaken in a time frame that is meaningful, as the timeliness of prediction is a key challenge for existing systems. This application scenario involves capture of data streams from wearable sensors onto a "hub" local to the patient (in the same house, for instance), and then periodic transmission of this data to a central repository for analysis. Three types of analysis may subsequently be undertaken on this data: (i) analysis and correlation of time series data from an individual patient (representing HbA1c concentration, blood pressure and blood lipid concentration for instance); (ii) analysis and correlation across a group of patients, indicating common trends; (iii) detection of templates within data to evaluate the occurrence of risk "signatures" within the recorded data. Data for a patient may be distributed across multiple repositories, or may involve data of different modalities (such as image data from MRI scans or retinopathy cameras, and numeric data from sensors). There are also significant constraints on how such data can be used and what trends can be reported – the use of privacy preserving data mining, especially over a distributed infrastructure, becomes important in this context.

In the same way as the Montage application described in section 3.1.1, we can list (Table 5) the autonomic adaptation mechanisms that could be applied to the HbPM application at the systems level. Data quality and secure access to patient information are two of the key application objectives in HbPM (Table 6). The requirement for these arises from the need to provide anonymized access to patient data, and ensuring that the level of security being offered can be modified based on the usage context (for instance, changing a 32-bit

Application Objective	**Autonomic Strategy**
Load Balancing	**1. Adapt task mapping granularity based on system capabilities/state** File staging, File splitting/merging Task rescheduling, Task migration File distribution and caching, Storage Management **2. Change fan-in/fan-out** DAG structure modification File staging, File splitting/merging Task rescheduling, Task migration File distribution and caching Storage Management
Handling Task Failure	**1. Reschedule the task on a different existing resource** File staging Task rescheduling, Task migration **2. Reschedule the task on a new resource** Resource discovery and allocation Task rescheduling File staging (migration/replication) **3. Roll back from checkpoint on the same resource** Checkpoint interval and granularity
Improving Throughput	**1. Increase fan out** Task rescheduling, Task migration File staging, File splitting/merging DAG structure modification File distribution and caching, Resource allocation **2. Change Scheduling Approach** File distribution (staging, merging, splitting, replication) Task rescheduling and mapping **3. Change Resource Mapping** Resource discovery, selection and allocation, File distribution (staging, merging, splitting, replication) Task rescheduling and mapping **4. Improve data access by re-indexing** Task rescheduling and mapping Data Caching

Table 4: ***Montage application management through Autonomic Strategies. Boldface Entries are the Strategies***

Vectors	Mechanisms
Coordination	Adapt Workflow structure
Communication	Message aggregation/buffering, Message replication, Caching, (value data) repeaters, Message re-transmission, Modification to message encryption Protocol
Execution Environment	Data persistence/replication, Sensor re-calibration, Task re-execution, Modification to data encryption mechanism, Resource selection and management, Workflow re-mapping and scheduling, Checkpoint/restart, Modification of anonymization algorithm

Table 5: ***Tuning Mechanisms in HbPM***

Application Objective	Autonomic Strategy
Ensuring Data Quality	**1. Re-calibrate sensors to improve data quality** Sensor re-calibration **2. Use robust messaging to improve data quality** Message re-transmission, Message replication, Caching, Buffering, Message repeaters
Modify Data Access Level	**1. Increased encryption of data to increase access control** Modify data encryption and/or Anonymization scheme **2. Modify authentication mechanism being used**
Modify Comms. Security Level	**1. Modify message encryption scheme and/or protocol used**

Table 6: ***HbPM application management through Autonomic Strategies***

encryption scheme to one which uses 128-bit encryption). Data quality primarily implies validating the quality of the data sources (patient sensors) and enable re-transmission of messages when missing values are detected. There are additional *application level* autonomic strategies that could also be applied to ensure data quality, but these would be very specific to this application, and therefore not generalizable to other scenarios and contexts.

3.3 Multi-Physics Applications: Nektar and Fusion

NEKTAR: Distributed Tightly-Coupled Fluid Dynamics [10]: The NEKTAR - Human Arterial Tree project is motivated by a grand challenge problem in bio-mechanics - simulation of blood flow in the entire human arterial network. Formation of arterial disease, such as atherosclerotic plaques, is strongly correlated to blood flow patterns, and is observed to occur preferentially in regions of separated and recirculating flows such as arterial branches and bifurcations. Interactions of blood flow in the human body can occur between different scales.

The challenge in modeling these types of interactions lies in the demand for large scale supercomputing resources to model the three-dimensional unsteady fluid dynamics within sites of interest such as arterial branches. What makes this type of model amenable to distributed computing is that the waveform coupling between the sites of interest can be reasonably modeled by a reduced set of one-dimensional equations, which capture the cross-sectional area and sectional velocity properties. One can therefore simulate the entire arterial tree using a hybrid approach based on a reduced set of one-dimensional equations for the overall system, and detailed 3D Navier-Stokes equations at arterial branches and bifurcations. The current human arterial tree model contains the 55 largest arteries in the human body with 27 arterial bifurcations. The inclusion of all 27 bifurcations in the simulation requires a total memory of around 7 TB, which is beyond the current capacity of any single supercomputing site available to the open research community in the US. This is a rare but interesting example of how a *traditional* application using a tightly-coupled application can be modified into a loosely-coupled distributed algorithm.

Using a hybrid approach to the arterial tree problem, NEKTAR simulates detailed 3D blood flow dynamics at arterial bifurcations while the waveform coupling between bifurcations is modeled through a reduced set of 1D equations. At each time step, the 1D results provide appropriate boundary conditions for the 3D simulations, such as the flow rate and pressure. This application represents a novel decomposition method. The parallel performance of the full-simulation does not scale efficiently beyond a thousand processors, hence devoting more resources to it does not result in a enhanced performance. But by decomposing the problem into separate 1D and 3D problems, this application is amenable to distribution, as typical data-transfers between the 1D-3D simulation components are much smaller, and thus internet scale communication does not lead to performance slowdown.

Coupled Fusion Simulation Application: The DoE SciDAC CPES fusion simulation project [11] is developing a new integrated Grid-based predictive plasma edge simulation capability to support next-generation burning plasma experiments, such as the International Thermonuclear Experimental Reactor (ITER). The typical application workflow for the project consists of coupled simulation codes, i.e., the edge turbulence particle-in-cell (PIC) code (GTC) and the microscopic MHD code (M3D), which run simultaneously on thousands of processors on separate HPC resources at supercomputing centers, and data must be steamed between these codes to achieve coupling. Furthermore, the data has to be processed en-route to the destination. For example, the data from the PIC codes has to be filtered through Ònoise detectionÓ processes before it can be coupled with the MHD code. As a result, effective online management and transfer of the simulation data is a critical part for this project and is essential to the scientific discovery process.

Although our discussion is focused on Nektar (Table 7), we find that several AOs as well as enabling mechanisms are common across a number of coupled multi-physics applications. One important AO for this class of application is "component synchronization" (Table 8). For an application which involves multiple, heterogenous components that are

Vectors	Mechanisms
Coordination	data-scheduling Co-scheduling
Communication	Message aggregation Message buffering, Message replication, Caching, Frequency of communication
Execution Environment	Data placement Task migration

Table 7: ***Tuning Mechanisms in Nektar***

Application Objective	Autonomic Strategy
Maintaining Component Concurrency	**Resource Management** Assign more/less computational resource Co-scheduling, data-scheduling **Change Granularity** Adapt frequency of communication data-scheduling
Scientific Fidelity	**Algorithmic Adaptivity** Change solvers -

Table 8: ***Nektar/multi-physics based application management through Autonomic Strategies***

coupled, a point of concern is that these different components will progress differently. Keeping these components synchronised is highly non-trivial. Specific mechanism that can be used vary from reassigning resources to the "slow" component, redistributing components. The reassignment in turn can have subsequent effects. We will refer to this strategy as "resource management". Another strategy for load-balancing is to alter the computation being performed, i.e., either slow down the fastest moving component that has blocked waiting for other components to provide data or change the granularity of the component that is slowing other components down. We will refer to this strategy as "change granularity"

Another AO of multi-physics simulations is the need to maintain *scientific fidelity*, which is often synonymous with accuracy or correctness. This in turn could require strategies such as "changing solvers" or "adaptivity" (eg mesh). In general there is a strict need to maintain the value of an operator (e.g. representing a conservation of some physical parameter, or the flux of a quantity). A specific strategy covering these mechanisms can be labelled as *algorithmic adaptivity*.

In case of the fusion simulations, a key requirement is that the data produced by one simulations must be streamed live to the other for coupling, as well as, possibly to remote sites for online simulation monitoring and control, data analysis and visualization, online validation, archiving, etc. The fundamental objective here is to efficiently and robustly stream data between live simulations or to remote applications so that it arrives at the destination just-in-time – if it arrives too early, times and resources will have to be wasted to buffer the data, and if it arrives to late, the application would waste resources waiting for the data to come in. A further objective is to opportunistically use in-transit resources to transform the data so that it is more suitable for consumption by the destination application, i.e., improve the quality of the data from the destination applications point of view. Key requirements and constraints include: (1) Enable high-throughput, low-latency data transfer to support near real-time access to the data; (2) Minimize overheads on the executing simulation; (3) Adapt to network conditions to maintain desired QoS – the network is a shared resource and the usage patterns typically vary constantly. (4) Handle network failures while eliminating loss of data – network failures usually lead to buffer overflows, and data has to be written to local disks to avoid loss, increasing the overhead on the simulation. (5) Effectively schedule and manage in-transit processing while satisfying the above requirements – this is particularly challenging due to the limited capabilities and resources and the dynamic capacities of the typically shared processing nodes.

The above objectives can be effectively achieved using autonomic behaviour [12, 13] using a range of strategies and mechanisms. The overall objective here is to maximize the utility of the data, both in terms of when it reaches the destination and whether it has been transformed in-transit to the form needed by the destination. These span (1) the application level, e.g., adapting solver behavior, adapting iteration count or using speculative computing by estimating the delayed data and rolling back if the estimation error exceeds a threshold; (2) the coordination level, e.g., adapting end-to-end and in-transit workflows as well as resources allocated to them; and (3) the data communication level, e.g., adaptive buffer management and adaptive data routing. Furthermore, this application also explored a hybrid autonomic approach that combined policy based autonomic managements with model-based online control [14].

4. ANALYZING AUTONOMIC APPLICATIONS

The discussion above clearly demonstrates how autonomic behaviors can be effectively applied to computational science applications and how applications can benefit from autonomics. Furthermore it can be seen that (1) both strategies and mechanisms cut across applications and (2) there is a many-to-many mapping between the strategies and mechanisms, i.e., the same mechanism may be used by multiple strategies and a strategy can use multiple mechanisms. The specific mechanisms used to implement a strategy depend upon the motivating Application Objective. Another important observation is that many mechanisms used by the autonomic strategies are already finding their way into existing applications are used in ad hoc ways to achieve adaptive behaviors. This former observation indicates that it is potentially feasible to envision library of reusable strategy templates, mechanisms and self-* patterns combining strategies and mechanism. The latter indicates that the transition to ACS is feasible and practical.

Often mechanisms can be similar for applications with similar values of the application vector, even though the specific application objective and strategy might be different. In other words, mechanisms that can be used to support different objectives using different strategies are clearly similar for different applications. As an example, see the communication vector for Nektar and HbPM. What this implies is that the exact autonomic behaviour and/or autonomic objective is often not critical, but the application characteristics and structure is – which is precisely what AV provide a handle too. This implies that mechanisms can and should be provided in a context neutral and infrastructure independent way. An analysis of applications also shows that autonomic objectives can be influenced via strategies that span across all vectors associated with an application.

Given that many different objectives can be implemented using similar mechanisms, it is worth considering a kind of autonomic "application similarity". This is a fine-grained similarity in that, applications that are characterized by the same mechanisms and strategy, irrespective of the specific objective, are considered similar. Similarly, applications with similar mechanisms and objectives, but different strategies can be considered similar. One can identify such similarity by looking along the vectors for specific types of mechanisms or objectives. For example, for the vectors addressing Communication, filestaging and messaging are mechanisms that exist for different objectives. Such fine-grained similarity arising due to a similar autonomic mechanism, should be distinguished from a coarse-grained similarity of autonomic applications, where the applications are essentially the same, and have high-level similarity in their functionality and goals. For instance, an application that processes audio files (compared to one that processes images) could have similar application objectives, and these could be achieved using similar mechanisms. Thus mechanisms developed for Montage could potentially be reused for digital audio applications.

5. CONCLUSION, DISCUSSION AND THE ROAD AHEAD

In this paper we have attempted to investigate the role of autonomics in Grid-based computational science application. We have taken an application perspective, and tried to understand for example, which autonomic behaviors and realizations are meaningful for a given application. We focus on three classes of existing applications (presented in Section 3), and using an understanding of their existing structure (in terms of execution units, communication and coordination mechanisms), propose how autonomic strategies could be developed based on them, and what application objectives could be achieved using them.

The focus in our analysis of these applications has been to explore the role and benefits of autonomics in Grid-based computational science applications, rather than being prescriptive in the steps that an application developer could follow to achieve it and build ACS applications. We believe that it is still somewhat premature to propose a prescriptive formalism that will be widely applicable, and as a result, this paper is an attempt to initiate a discussion that could potentially lead to such a formalism. In contrast, significant current efforts, especially from providers of autonomic toolkits and products (such as "ABLE" from IBM), have focused on models and tools that can be used to support autonomic tuning. However, we believe that, due to the nature of Grid-based computational science applications it is necessary to take a more application-centric perspective, to better understand why and at where the use of autonomics can be effectively used.

Hence, the aim of this paper has been to raise more questions than to answer them, with the hope of laying out the motivations, issues, challenges and potential benefits that frame the landscape of Grid-based autonomic computational science. Having done so, we hope to eventually provide in future work answers to the many questions, and solutions to the challenges that lie ahead.

Ultimately, what we aim to understand is how do we develop autonomic computational scientific applications to efficiently and effectively utilize distributed resources? What are the challenges in developing such autonomic applications? Interestingly, many of the challenges that general purpose distributed applications face (listed in Ref [2]) are common to autonomic applications. For example, a self-* pattern can be considered a recurring ordering or grouping of mechanisms that together achieve an autonomic strategy. In other words, self-* patterns identify mappings from mechanisms into autonomic strategies. Identifying such self-* patterns, and their subsequent support will facilitate the realization of autonomic applications and the construction of autonomic strategies from these mechanisms. Once supported, we can investigate the applicability of different self-* patterns for a range of ACS applications as well as Grid infrastructures.

Based on the investigation and analysis in this paper, we can identify the following challenges in the development and deployment of autonomic applications:

- From figure 1 identifying the best way to tune an application "behaviour" is a difficult and time consuming process. Significant current focus has been on *automating* application or data partitioning (i.e. a very specific behaviour), for instance. However, extracting greater benefit from autonomic computing strategies remains a difficult challenge to realize. This stems from: (i) the inability of an end user (e.g. a scientist) to adequately express tuning policies that be applied without user intervention; (ii) the unavailability of suitable alternative components (e.g. numeric solvers) that can be replaced by another in a meaningful manner (and still lead to accurate results).

- From section 3, it is useful to note that there are common mechanisms within an autonomic strategy used to achieve a particular application objective. For instance, *file staging* recurs in various autonomic strategies in the Montage application. An important challenge relates to how such common mechanisms can be supported in software libraries to accelerate the development of autonomic applications.

- Identifying suitable monitoring strategies that could facilitate autonomic adaptation also remains an important challenge. For instance, identifying what (i.e. which part of the application) and how often to monitor is a subjective decision made by the autonomic systems developer. The capability to support an *active* monitoring regime, whereby a tuning engine decides that additional data capture would be useful to improve its model of the application (and thereby support tuning), would provide significant benefit over current approaches. For instance, the tuning engine can change the interval at which data is collected if there is too much data to analyse, or the analysis itself on q large data set imposes a high overhead. On the other hand, the tuning engine may decide to increase the monitoring interval if it notices that it gets a very large error when attempting to model the recorded data – and additional precision is necessary. In both instances, the idea is to use the monitoring interval itself as a tunable parameter.

- Autonomic tuning could also be used in an exploratory context, to investigate possible input parameter combinations that reveal *interesting* behaviours to an end user. Understanding how parameter search space could be better explored through autonomic strategies also remains an important future challenge.
- Additionally, there remains the issue of relinquishing control and decision making to the Autonomic system and manager, and the circumstances under which developers will generally agree to do so. Simple benchmarks and performance measures will be required. This in turn required some level of standardization using monitoring.
- Finally revisiting the issue of application similiarity, it will be useful to understand why and under what conditions strategies differ when application vectors are similar for applications?

Acknowledgements

This paper is the outcome of the e-Science Institute sponsored Research Theme on Distributed Programming Abstractions. We would like to thank Murray Cole, Daniel Katz and Jon Weissman for being partners in the DPA expedition and having contributed immensely to our insight and understanding of distributed applications and systems, which form the basis of their application specifically to Autonomic Computational Science.

6. REFERENCES

[1] Autonomic Grid Computing – Concepts, Requirements, Infrastructures, M. Parashar, Autonomic Computing: Concepts, Infrastructure and Applications, Editors: M. Parashar and S. Hariri, CRC Press, 2006.

[2] Shantenu Jha, Murray Cole, Daniel Katz, Manish Parashar, Omer Rana, and Jon Weissman. Abstractions for large-scale distributed applications and systems. *ACM Computing Surveys*, 2009. *under review.*

[3] Salim Hariri, Bithika Khargharia, Houping Chen, Jingmei Yang, Yeliang Zhang, Manish Parashar, and Hua Liu. The autonomic computing paradigm. *Cluster Computing*, 9(1):5–17, 2006.

[4] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[5] S. Chandra and M. Parashar. Addressing Spatiotemporal and Computational Heterogeneity in Structured Adaptive Mesh Refinement. *Journal of Computing and Visualization in Science*, 9(3):145–163, 2006.

[6] Rashid J. Al-Ali, Kaizar Amin, Gregor von Laszewski, Omer F. Rana, David W. Walker, Mihael Hategan, and Nestor J. Zaluzec. Analysis and Provision of QoS for Distributed Grid Applications. *Journal of Grid Computing (Springer)*, 2(2):163–182, 2004.

[7] J. C. Jacob, D. S. Katz, G. B. Berriman, J. Good, A. C. Laity, E. Deelman, C. Kesselman, G. Singh, M.-H. Su, T. A. Prince, and R. Williams. Montage: A grid portal and software toolkit for science-grade astronomical image mosaicking. *International Journal of Computational Science and Engineering*, 3, 2007.

[8] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahl, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming Journal*, 13(3):219–237, 2005.

[9] J. Frey, T. Tannenbaum, M. Livny, and S. Tuecke. Condor-G: a computation management agent for multi-institutional grids. In *Proceedings of the 10th IEEE Symposium on High-Performance Distributed Computing*, 2001.

[10] NEKTAR, SPICE and Vortonics: Using Federated Grids for Large Scale Scientific Applications, Cluster Computing, Vol. 10, No. 3, 351-364 (2007,) DOI 10.1007/s10586-007-0029-4.

[11] S. Klasky, M. Beck, V. Bhat, E. Feibush, B. Ludäscher, M. Parashar, A. Shoshani, D. Silver, and M. Vouk. Data management on the fusion computational pipeline. *Journal of Physics: Conference Series*, 16:510–520, 2005.

[12] V. Bhat, M. Parashar, M. Khandekar, N. Kandasamy, and S. Klasky. A Self-Managing Wide-Area Data Streaming Service using Model-based Online Control. In *7th IEEE International Conference on Grid Computing (Grid 2006)*, pages 176–183, Barcelona, Spain, 2006. IEEE Computer Society.

[13] V. Bhat, M. Parashar, and S. Klasky. Experiments with In-Transit Processing for Data Intensive Grid workflows. In *8th IEEE International Conference on Grid Computing (Grid 2007)*, pages 193–200, Austin, TX, USA, 2007. IEEE Computer Society.

[14] V. Bhat, M. Parashar, H. Liu, M. Khandekar, N. Kandasamy, and S. Abdelwahed. Enabling Self-Managing Applications using Model-based Online Control Strategies. In *3rd IEEE International Conference on Autonomic Computing*, pages 15–24, Dublin, Ireland, 2006.