CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2011; **23**:1990–2019
Published online 23 May 2011 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.1734

A taxonomy and survey on autonomic management of applications in grid computing environments

Mustafizur Rahman[1], Rajiv Ranjan[2], Rajkumar Buyya[1,*,†] and Boualem Benatallah[2]

[1]*Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Australia*
[2]*Service Oriented Computing Research Group, School of Computer Science and Engineering, The University of New South Wales, Australia*

SUMMARY

In Grid computing environments, the availability, performance, and state of resources, applications, services, and data undergo continuous changes during the life cycle of an application. Uncertainty is a fact in Grid environments, which is triggered by multiple factors, including: (1) failures, (2) dynamism, (3) incomplete global knowledge, and (4) heterogeneity. Unfortunately, the existing Grid management methods, tools, and application composition techniques are inadequate to handle these resource, application and environment behaviors. The aforementioned characteristics impose serious requirements on the Grid programming and runtime systems if they wish to deliver efficient performance to scientific and commercial applications. To overcome the above challenges, the Grid programming and runtime systems must become autonomic or self-managing in accordance with the high-level behavior specified by system administrators. Autonomic systems are inspired by biological systems that deal with similar challenges of complexity, dynamism, heterogeneity, and uncertainty. To this end, we propose a comprehensive taxonomy that characterizes and classifies different software components and high-level methods that are required for autonomic management of applications in Grids. We also survey several representative Grid computing systems that have been developed by various leading research groups in the academia and industry. The taxonomy not only highlights the similarities and differences of state-of-the-art technologies utilized in autonomic application management from the perspective of Grid computing, but also identifies the areas that require further research initiatives. We believe that this taxonomy and its mapping to relevant systems would be highly useful for academic- and industry-based researchers, who are engaged in the design of Autonomic Grid and more recently, Cloud computing systems.
Copyright © 2011 John Wiley & Sons, Ltd.

Received 17 November 2010; Accepted 21 February 2011

KEY WORDS: grid computing; workflow management; autonomic systems

1. INTRODUCTION

Many Scientific discoveries are increasingly being made through collaborations. This is mainly due to the non-availability of all required resources (knowledge, information, hardware, software) within a single organization (universities, government institutions, business enterprises). Further, it is often not viable to maintain and administer all the required resources within a single organization, due to economic factors and infrequent usage patterns. In the past few years, many countries including Australia have launched ambitious programs that facilitate the creation of infrastructures

*Correspondence to: Rajkumar Buyya, Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Australia.
†E-mail: raj@csse.unimelb.edu.au

Copyright © 2011 John Wiley & Sons, Ltd.

that aim to perform large-scale scientific experiments. Collectively, such programs are referred to as e-Science in the UK, Grids in Europe, and Asia, Cyber Infrastructure in the US, and e-Research in Australia.

Over the last decade, Grids (TeraGrid, ChinaGrid, UK eResearch Grid, Australian Grid, Core-Grid) have established itself as the distributed and collaborative resource-sharing environment for hosting various large-scale scientific applications, such as eResearch workflows, financial simulations. Thus, runtime systems and techniques for managing applications have emerged as one of the most important Grid services in past few years. An Application Management System (AMS) is generally employed to compose, define, deploy, and execute these scientific applications on Grid resources (computational clusters, supercomputers). However, the increasing scale, complexity, heterogeneity and dynamism of Grid computing environments (networks, resources and applications) have made such AMSs brittle, unmanageable and insecure.

In Grid computing environments, the availability, performance, and state of resources, applications, services, and data undergo continuous changes during the life cycle of an application. Uncertainty is a fact in Grid environments, which is triggered by multiple factors, including: (1) failures as the system and application scales, which accounts to severe performance degradation, (2) dynamism, which occurs due to temporal behaviors that should be detected and resolved at runtime, (3) incomplete global knowledge that leads to problem of non-coordinated decision making for using the resources and network bandwidth across the infrastructure, and (4) heterogeneity that occurs due to availability of different types of Grid resources, network architectures, and access policies. Unfortunately, the current Grid programming, runtime systems, and application composition techniques are inadequate to handle these resource, application, and environment behaviors. The aforementioned characteristics impose serious requirements on the services, programming, and runtime systems support for scientific and commercial applications in Grid environments.

The above requirements indicate that Grid programming and runtime systems must be able to support the following behaviors: (1) ability to handle the spike in demand across the organization through dynamic scaling-in of resources from the massive resource pool; (2) dynamically adapt to performance, failure, leave, join of hardware and software including resources, softwares, applications, and networks; (3) be able to coordinate activities (scheduling, allocation, recovery) with others in the system; and (4) handle the scale of the system while ensuring a secure and cohesive environment. Furthermore, the core services at infrastructure level must address the lack of reliability, uncertainty, and dynamism (leave, join, failure) of the execution platform and at the same time provide support for decentralized and deterministic resource discovery and monitoring.

Autonomic Computing (AC) [1] is an emerging area of research for developing large-scale, self-managing, complex distributed (Grids, Clouds, Data Centers) system. The vision of AC is to apply the principles of self-regulation and complexity hiding for designing complex computer-based systems. Thus, AC provides a holistic approach for the development of systems/applications that can adapt themselves to meet requirements of performance, fault tolerance, reliability, security, Quality of Service (QoS), etc., without manual intervention. An autonomic Grid system leverages the concept of AC and is able to efficiently define, manage, and execute applications in heterogeneous and dynamic Grid environment by continuously adapting itself to the current state of the system.

1.1. Our contributions

The concrete contributions made by this paper include:

- A detailed survey of existing Grid systems that aim at autonomic management of applications.
- Characterization of above systems with respect to proposed taxonomy such as application composition, scheduling, monitoring, coordinating, and failure handling as well as how the self-management properties (self-configuring, self-optimizing, self-healing, and self-protecting) have been supported by these systems.
- Discussion of the future research challenges related to autonomic application management in Grid and more recently, Cloud computing environments.

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

1.2. Paper organization

The remainder of the paper is organized as follows. Section 2 presents the history of AC and the path along which this concept is evolved. Section 3 describes an overview of ACS and its properties. The concepts of autonomic application management in Grid computing environment are discussed in Section 4. We propose the taxonomy in Section 5 that categorizes autonomic application management with respect to key features of AC. Section 6 provides a detailed survey of few selected Grid systems and mapping of the proposed taxonomy onto these systems. In Section 7, we present a brief discussion and identification of potential research opportunities in this area. We end this paper with some final remarks in Section 8.

2. HISTORY OF AC

AC is a self-managing computing model and the word autonomic is derived from its biological origins. The control in the human body works in such a manner (self-regulating) that usually no humans' interference and consciousness are required. Likewise, the goal of AC is to create systems that run themselves, capable of high-level functioning while keeping the system's complexity invisible to the user. In this section, we briefly illustrate the introduction of AC in the field of Information Technology and describe its evolution along the path of the twenty-first century's technological revolution.

2.1. Integrating biology and information technology

The term AC is named after and patterned on the human body's Autonomic Nervous System (ANS) [1]. ANS is the cornerstone to our ability to perceive, adapt to, and interact with the world around us; thus, helping human beings to manage dynamically changing and unpredictable circumstances. It acts as a control system functioning largely below the level of consciousness and handles the human body's management of breathing, digestion, salivation, fending off germs and viruses, etc. as shown in Figure 1.

Inspired by the functionalities of ANS, AC has emerged to equip computing systems with the self-managing mechanisms of the human body achieved through ANS. An Autonomic Computing System (ACS) manages and control the functioning of computing systems and applications without any user input or intervention, in the same way as ANS regulates the human body systems without conscious input from the individual. Similar to ANS, ACS constantly checks and monitors its external and internal environment and automatically adapts to changing conditions in order to manage, optimize, repair, and protect itself.

2.2. Evolution of AC

In order to address the growing heterogeneity, complexity, and demand of computer systems, researchers of several organizations took initiatives to develop autonomous and self-managing systems in the early 1990s and it continued throughout the whole decade. These research initiatives gradually became matured and eventually facilitated AC to emerge as an area of research.

Similar to the birth of Internet, one of the notable preliminary self-managing projects has been initiated by the Defense Advanced Research Projects Agency (DARPA) in 1997 for a military application [2]. The project was called Situational Awareness System (SAS) and its aim was to create personal communication and location devices for soldiers in the battlefield. As an outcome, soldiers had been able to enter the status report (i.e. discovery of enemy tanks) into their personal device. This information would automatically spread to all other soldiers based on a decentralized peer-to-peer (P2P) mobile adaptive routing. The latest status report could then be called up accordingly, while entering an enemy area.

Further, DARPA initiated another project related to self-management, named Dynamic Assembly for Systems Adaptability, Dependability, and Assurance (DASADA). The objective of the DASADA program was to research and develop technology that would enable mission critical systems to

Copyright © 2011 John Wiley & Sons, Ltd.
DOI: 10.1002/cpe

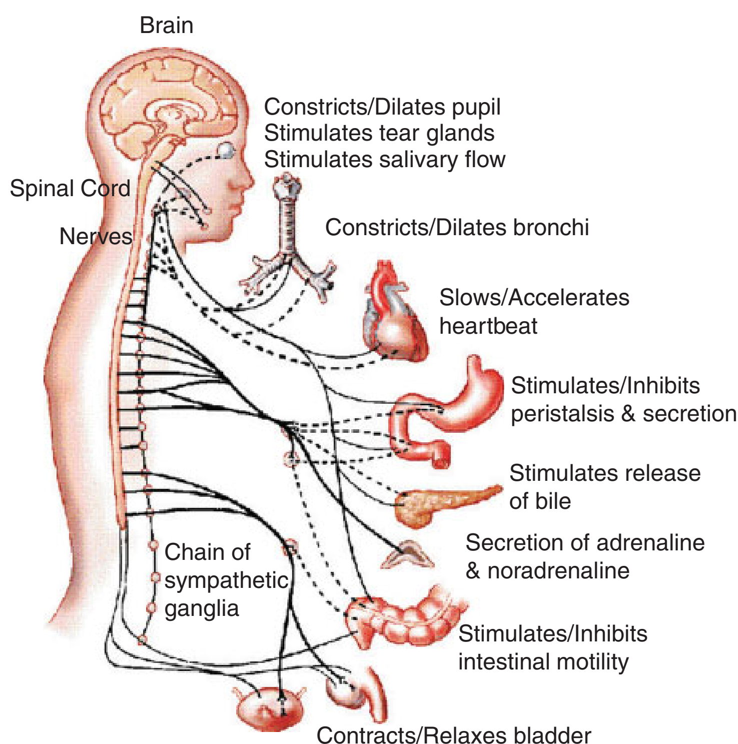

Figure 1. Autonomic nervous system of human body.

meet high assurance, dependability, and adaptability requirements. Essentially, it deals with the complexity of large distributed software systems and pioneered the architecture-driven approach to self-management [3].

NASA utilized the features of autonomous systems in the late 1990s for its space projects such as DS1 (Deep Space 1) and Mars Pathfinder [4]. In particular, NASA's interest was to make its deep-space probes more autonomous so that the probes can quickly adapt to extraordinary situations. To address this challenge, NASA designed a Remote Agent architecture, where the Remote Agent integrates constraint-based temporal planning [5] and scheduling, robust multi-threaded execution, and model-based mode identification and reconfiguration. Thus, spacecrafts were able to carry out autonomous operations for long periods of time with no human intervention.

IBM started the AC initiative in 2001 with the ultimate aim to develop computing systems capable of self-management so that it can overcome the rapidly growing complexity of computing systems management [6]. On March 8, 2001, IBM Senior Vice President and Director of Research Dr Paul Horn presented the importance and direction of AC during a keynote speech in the National Academy of Engineering conference at Harvard University [7]. He suggested that complex computing systems should be able to independently take care of the regular maintenance and optimization tasks; thus, reducing the workload on the system administrators. Shortly after, IBM Server Group introduced the Server's Group project with codename eLiza). Eventually, *Project eLiza* became known as the AC project. Thus, began the AC journey within IBM.

In 2003, an IBM introduced architectural blueprint to build ACS [8]. In that blueprint, IBM proposed the architectural concept of AC and described five building blocks for an autonomic system. It also outlined the four properties of an autonomic (i.e. self-managing) system namely, self-configuring, self-optimizing, self-healing, and self-protecting. These properties are described in detail in Section 3.1.

Gradually, IBM became the leader in the AC space and offered many effective self-managing software toolkits such as Tivoli. In 2005, IBM launched a set of new development tools for AC with the hope that these will help to pave the way for increased mainstream adoption of AC. The

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

Table I. Timeline of autonomic computing evolution.

Year	Term	Description
1997	Situational Awareness System (SAS)	DARPA initiated SAS project to provide the soldiers real-time situational awareness information
1998	Autonomous Agent	NASA made its deep-space probes more autonomous
2000	DASADA	DARPA introduced gauges and probes in the architecture of software systems for monitoring the system
2001	Autonomic Computing	IBM pushed Autonomic Computing
2003	AC Blueprint	IBM introduced architectural blueprint to build autonomic computing system
2005	AC Development Tools	IBM offered new development tools for autonomic computing.
2005	Autonomic Grid Computing	The concept of autonomic Grid computing was proposed
2009	Autonomic Cloud Computing	Principles of autonomic computing was utilized in computational Clouds

release included an autonomic management tool called, Policy Management for AC (PMAC) that makes decisions based on policies or business rules created by the developers when embedded within software applications.

In 2005, Parashar *et al.* [9] addressed the constant growth and increasing scale complexity of dynamic and heterogeneous components in computational Grids, and proposed to utilize the AC features for the composition, deployment, and management of complex applications in such an environment. As a part of this initiative, they introduced project Automate [10], which provides a framework for enabling autonomic application management in Grids.

Recently, the world has seen a paradigm shift in the consumption and delivery of IT services with the emergence of Cloud Computing. The computational Clouds address the explosive growth of internet-wide computational/storage devices, and provides a superior user experience through scalability, reliability, and utility. The principles of AC have also been utilized in computational Clouds [11].

The emergence and evolution of AC from its origin, autonomous systems can be realized in brief from Table I.

3. OVERVIEW OF ACSs

An ACS makes decisions on its own using high-level policies in order to achieve a set of goals. It constantly checks, monitors, and optimizes its status, and automatically adapts itself to the changing conditions. As widely reported in the literature [8], an ACS is composed of Autonomic Elements (AE) interacting with each other. AE is the basic building block of ACS and can be considered as a software agent. An AE consists of one Autonomic Manager (AM) and one or more Managed Element (ME). The core component of AE is a control loop that integrates AM with ME (refer to Figure 2). The functionality of this control loop is similar to the generic agent model proposed by Russell and Norvig [12], in which, an intelligent agent perceives its environment through sensors and uses these percepts to determine actions to execute on the environment.

The Managed Element is a software or hardware component from the system, which is given autonomic behavior by coupling it with an AM. Thus, ME can be a web server or database, a specific software component in an application (e.g. the query optimizer in a database), the operating system, a cluster of machines in a Grid environment, a stack of hard drives, a wired or wireless network, a CPU or printer, etc.

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

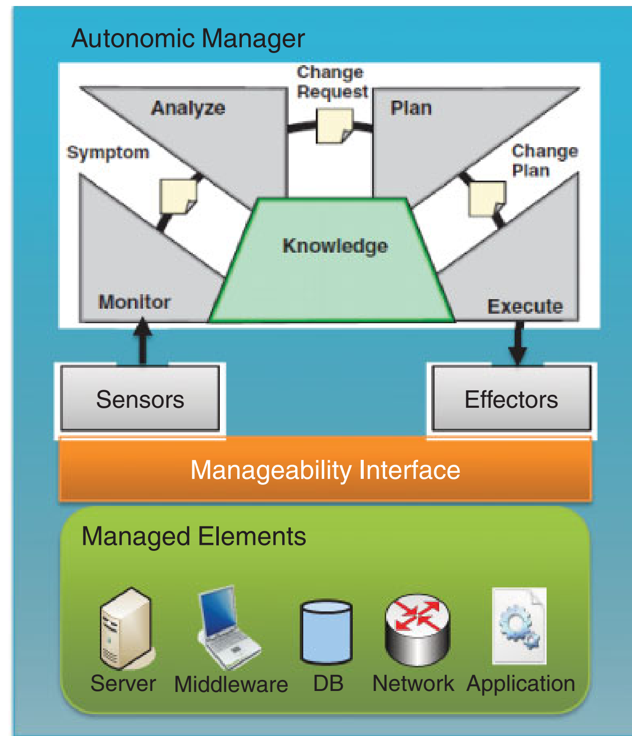

Figure 2. Architecture of an autonomic element.

The autonomic manager is a software component that can be configured by system administrators using high-level goals. It uses the monitored data from sensors and internal knowledge (i.e. rules) of the system to plan and execute the low-level actions that are necessary to achieve these goals. The goals are usually expressed by event-condition-action policies (e.g. when 95% of Web servers' response time exceeds 2 s and there are available resources, then increasing number of active Web servers) or utility function policies (e.g. increasing and distributing available resources among different serves so that the utility is maximized).

AM uses a manageability interface (i.e. sensors and effectors) to monitor and control MEs and a five component analysis and planning engine (comprising of Monitor, Analysis, Plan, Execute, and Knowledge base) to manage self-managing activities. Sensors retrieve information regarding the current state of ME and Effectors execute the required actions setup by AM. Thus, sensors and effectors are linked together to create the control loop.

The Monitor observes the Sensors, filters the data or system information collected by the sensors (network/storage usage or CPU/memory utilization) and stores the relevant data in the Knowledge base. The Analysis engine compares the gathered data against the desired or expected values stored in the Knowledge base. The Planning engine devises strategies to correct or adjust the trends identified by the Analysis engine. The Execution engine finally carries out changes (e.g. adding/removing servers to a Web server cluster or changing configuration parameters in a Web server) to the ME through Effectors and stores the affected values in the Knowledge base.

3.1. Properties of ACS

ACSs are generally composed of AEs and capable of managing their behaviors and relationships with other systems in accordance with high-level policies. An autonomic system should possess at least eight key properties or characteristics. The primary four properties (refer to Figure 3) of an autonomic system are called self-* properties, which are:

1. *Self-configuring*: An autonomic system must be able to automatically configure (i.e. setup) and reconfigure itself under dynamic and changing conditions.

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

Figure 3. Self-* properties of autonomic computing system.

Table II. Summary of self-* properties.

Self-* Property	Description	Example
Self-configuring	Ability to adapt to changes in the system	Installing software when it detects that some prerequisite software components are missing
Self-optimizing	Ability to improve performance of the system	Adjusting the current workload when it observes an increase or decrease in capacity
Self-healing	Ability to discover, diagnose and recover from faults	Automatically re-indexing the files if a database index fails
Self-protecting	Ability to anticipate, detect, identify and protect against threats/intrusions	Taking resources offline if it detects an intrusion attempt

2. *Self-optimizing*: An autonomic system must be able to optimize its working by monitoring the status quo and taking appropriate actions.
3. *Self-healing*: An autonomic system must be capable of identifying potential problems/failures and recovering from unexpected events that might lead the system to malfunction.
4. *Self-protecting*: An autonomic system must be capable of detecting and protecting itself from malicious attacks so as to maintain overall system security and integrity.

The secondary properties of autonomic systems are:

1. *Self-awareness*: An autonomic system requires to *know itself*, which can be achieved by having a detailed knowledge of its components and connections with other systems.
2. *Context-awareness*: An autonomic system should be aware of its execution environment by exposing itself and discovering other AEs or systems in the environment.
3. *Openness*: An autonomic system should be able to function in a heterogeneous environment and be implemented on open standards and protocols.
4. *Anticipatory*: One critical property from the perspective of the users is that an autonomic system should be able to anticipate its needs and behaviors to act accordingly, while keeping its complexity hidden.

The description and example of the primary four properties of an autonomic system are presented in Table II.

3.2. Example implementation of AC: IBM tivoli

Tivoli [13] is a Systems Management Software toolkit provided by IBM that enables automation of routine management tasks for individual resource elements. It is designed based on CORBA-based

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

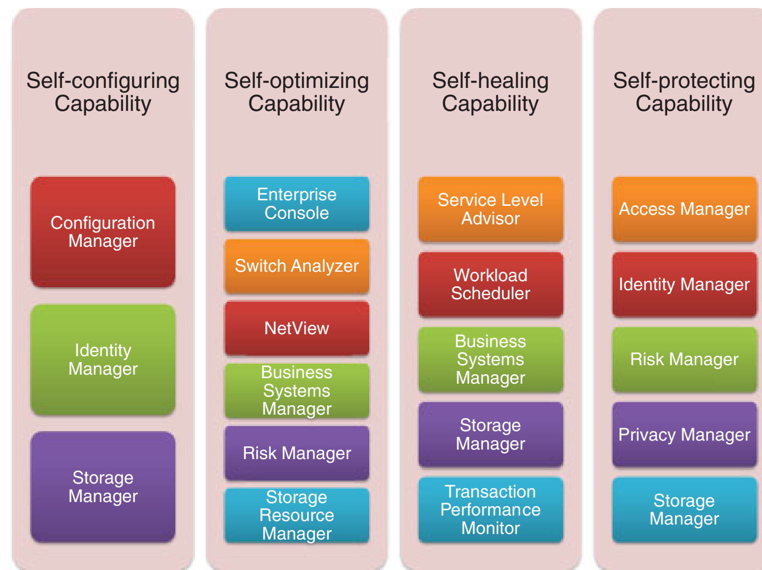

Figure 4. Components of IBM Tivoli enterprise management software toolkit.

architecture, which allows Tivoli to manage a large number of remote locations. The Tivoli software tools are focused on various aspects of system management (e.g. security, storage, performance, availability, configuration, and operations) as well as facilitate provisioning of a wide range of resources including systems, applications, middleware, networks, and storage devices.

The purpose of the Tivoli platform is to bring self-managing capabilities into the IT infrastructure (Figure 4). The Tivoli software availability management portfolio provides tools to help customers monitor the health and performance of their IT infrastructure. The Tivoli storage management tools help users to automatically and efficiently back up and protect data. The workload management tools use self-optimizing technology to optimize hardware and software use and verify that SLA goals are met successfully. Monitoring and event correlation tools help to determine when changes in the IT infrastructure require reconfiguration actions. These tools can allow users to reconfigure their IT environment within minutes or hours rather than in days or weeks. The self-managing capabilities of the Tivoli software toolkit are discussed in the following.

3.2.1. Self-configuring capabilities. In order to implement a self-configuring environment, Tivoli uses three software components: Configuration Manager, Storage Manager, and Identity Manager. The Tivoli Configuration Manager automatically configures to rapidly changing environments. It provides an inventory scanning engine and a state management engine that can sense and detect when software on a target machine is out-of-synchronization with respect to a reference model for that class of machine. The Tivoli Storage Manager provides self-configuring capabilities by automatically identifying and loading the appropriate drivers for the storage devices connected to the server. The Tivoli Identity Manager uses automated role-based provisioning for dynamic account creation for users.

3.2.2. Self-optimizing capabilities. The Tivoli Service Level Advisor performs self-optimizing activity by preventing Service Level Agreement (SLA) breaches with predictive capabilities. Based on the analysis of historical performance data from the Tivoli Enterprise Data Warehouse, it can predict when critical SLA thresholds could be exceeded in the future. The Tivoli Workload Scheduler monitors and controls the flow of work through the IT infrastructure, and uses sophisticated algorithms to maximize throughput and optimize resource usage. The Tivoli Storage Manager supports Adaptive Differencing technology that facilitates the backup archive client to dynamically determine efficient approaches for creating backup copies of just changed bytes, blocks or

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

files, and delivering improved backup performance. The Tivoli Business Systems Manager enables optimization of IT problem repairs based on the business impact of outages.

3.2.3. Self-healing capabilities. Tivoli utilizes several tools for implementing a self-healing environment. The Tivoli Enterprise Console collects and compares error reports, derives root cause, and initiates corrective actions. The Tivoli Switch Analyzer correlates network device (Layer 2 switch) errors to the root cause without user intervention. The Tivoli NetView enables self-healing by discovering TCP/IP networks, displaying network topologies, monitoring network health, and gathering performance data. The Tivoli Storage Resource Manager automatically identifies potential problems through scanning and executes policy-based actions to resolve allocation of storage quotas or space and provide application availability.

3.2.4. Self-protecting capabilities. The Tivoli Storage Manager self-protects by automating backup and archival of enterprise data across heterogeneous storage environments. On the other hand, the Tivoli Access Manage self-protects by preventing unauthorized access and using a single security policy server to enforce security across multiple file types, applications, devices, operating systems, and protocols. The Tivoli Risk Manager enables self-protecting by assessing potential security threats and automating responses, such as server reconfiguration, security patch deployment, and account revocation.

4. AUTONOMIC APPLICATION MANAGEMENT IN GRIDS

Computational Grids enable the sharing, selection, and aggregation of geographically distributed heterogeneous resources, such as computational clusters, supercomputers, storage devices, and scientific instruments. These resources are under control of different Grid sites being utilized to solve many important scientific, engineering, and business problems. In general, Grid infrastructures are distributed, large, heterogeneous, uncertain, and highly dynamic. A sample scenario of scientific application composition and management in such Grid environment is shown in Figure 5, where

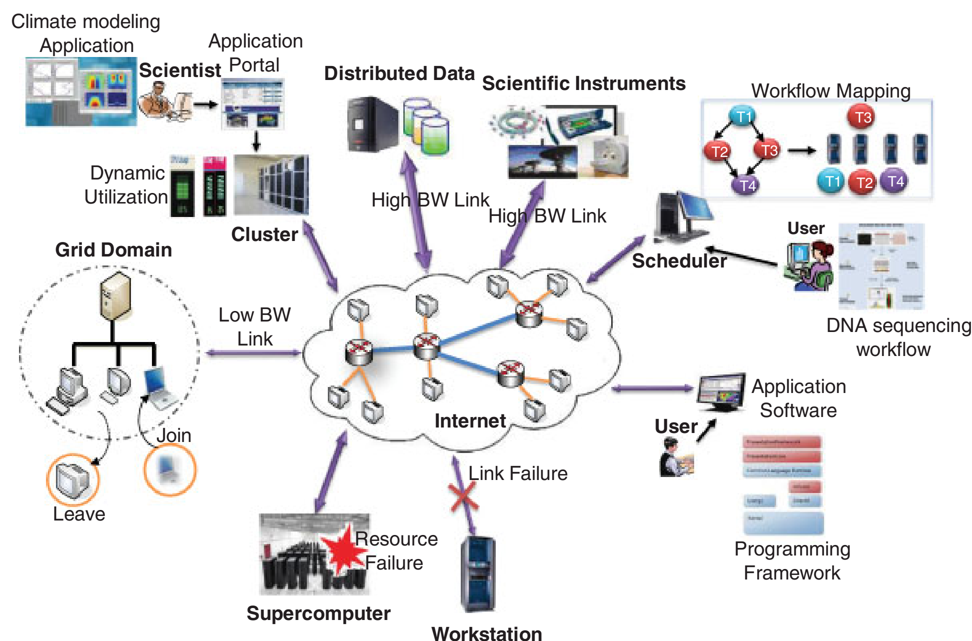

Figure 5. Heterogeneous and dynamic Grid computing environment.

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

application scientists and users from different Grid sites/domain share various types of resources distributed world wide.

Application scientists/users are not much concerned about the low-level Grid infrastructure, such as runtime systems, programming environment and core middleware services, but still want high confidence level in composition and deployment of their scientific applications. They rely on the third-party brokering or scheduling services, which abstract the underlying complexity of the system and facilitates efficient execution of the applications in Grid resources. However, many important applications in bioinformatics, medical imaging, and data mining require very accurate and reliable tools for conducting distributed experiments and analysis. The traditional Grid management methods, tools, and application composition techniques are inadequate to handle the dynamic interaction between the components and the sheer scale, complexity, uncertainty, and heterogeneity of the Grid infrastructures as shown in Figure 5.

AC has emerged to cope with the aforementioned challenges by being decentralized, context aware, adaptive, and resilient. Thus autonomic application management in Grids implies the utilization of AC features for the composition, deployment, and management of complex applications in Grids.

5. TAXONOMY

In this section, we propose a taxonomy that categorizes and classifies the approaches of autonomic application management in the context of computational Grids with respect to the key features of AC. As shown in Figure 6, it consists of six elements of autonomic application management: (1) application composition, (2) application scheduling, (3) coordination, (4) monitoring, (5) self-* property, and (6) system characteristics. In this section, we discuss each element and its classification in detail.

5.1. Application composition

The applications executed in a distributed computing environment such as Grids are generally computation or data intensive and users can experience better performance if they are able to execute these applications in parallel. In order to facilitate autonomic application management, the applications are needed to be composed dynamically based on the system configuration and users' requirements. As shown in Figure 7, the application composition in a computational Grid can be characterized by four factors: (a) application type, (b) application domain, (c) application definition, and (d) data requirement.

5.1.1. Application type. An application is composed of multiple tasks, where a task is a set of instructions that can be executed on a single processing element of a computing resource. Based on the dependency or relationship among these tasks, Grid applications can be divided into three types: Bag-of-task (BOT), Message Passing Interface (MPI), and Workflow.

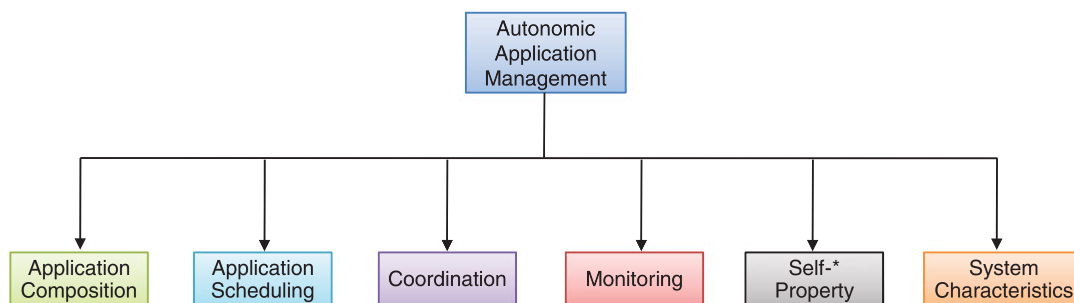

Figure 6. Elements of autonomic application management.

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

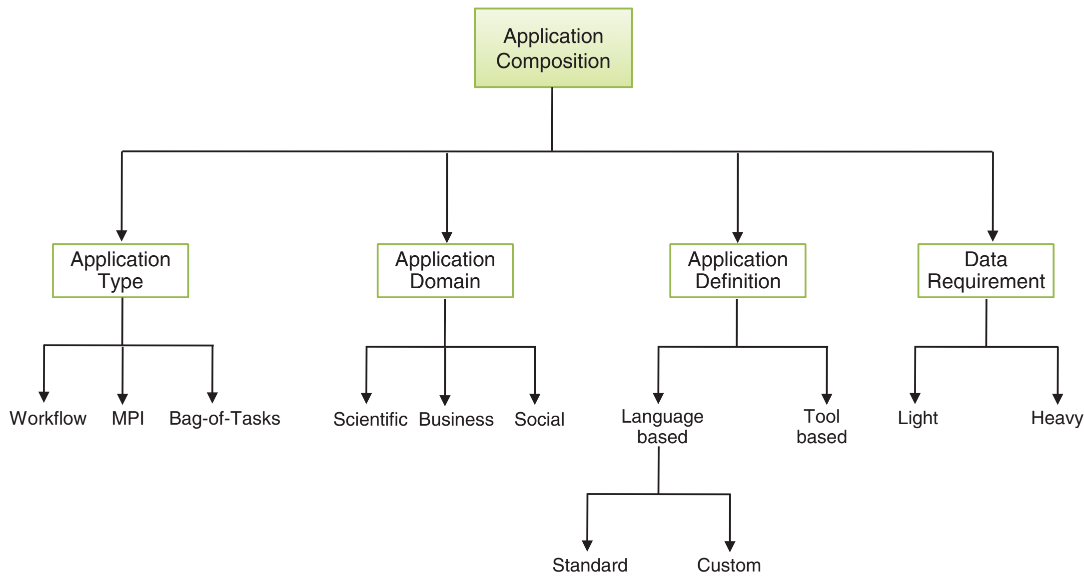

Figure 7. Application composition taxonomy.

A BOT application [14] consists of multiple independent tasks with no communication among each other. The final result or output of executing the BOT application is achieved once all these tasks are completed. On the other hand, MPI applications [15] are composed of multiple tasks where inter-task communication is developed with MPI. Since the tasks in most MPI applications need to communicate with each other during execution, the necessary processing elements are required to be available at the same time to minimize application completion time.

Finally, a workflow application can be modeled as a Directed Acyclic Graph (DAG), where the tasks in the workflow are represented as nodes in the graph and the dependencies among the tasks are represented as the directed arcs among the nodes [16]. In a workflow, a task that does not have any parent task is called an entry task and a task that does not have any child task is called an exit task. A child task cannot be executed until all of its parent tasks are completed. The output of a workflow application is achieved when the exit tasks finish execution.

5.1.2. Application domain. Grids offer a way to solve challenging problems by providing a massive computational resource-sharing environment of large-scale, heterogeneous, and distributed IT resources.With the advent of Grid technologies, scientists and engineers are building more and more complex applications to manage and process large-scale experiments. These applications are spanned across three domains: scientific, business, and social.

Many scientific (also known as e-Science and e-Research) applications such as Bioinformatics, Drug discovery, Data mining, High-energy physics, Astronomy, and Neuroscience have been benefited with the emergence of Grids. Enabling Grids for E-sciencE (EGEE) [17] is considered as one of the biggest initiatives taken by European Union to utilize Grid technologies for scientific applications. Likewise, Business Experiments in GRID (BEinGRID) [18] is also the largest project for facilitating business applications such as Business process modeling, Financial modeling, and forecasting using Grid solutions. Recently, the emergence and upward growth of various social applications such as Social networking have been widely recognized and the scalability of distributed computing environment is being leveraged for the better performance of these type of applications.

5.1.3. Application definition. In general, users can define applications using definition languages or tools. In terms of definition language, markup language such as Extensible Markup Language (XML) [19] is widely used specially for workflow specification as it facilitates information description in a nested structure. Therefore, many XML-based application definition languages

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

have been adopted in Grids. Some of these languages such as WSDL [20], BPEL [21] have been standardized by the industry and research community (i.e. W3C [22]), whereas some of them are customized xWFL [23], AGWL [24] according to the requirements of the system.

Although language-based definition of applications is convenient for expert users, it requires users to memorize a lot of language-specific syntax. Thus, the general users prefer to use Graphical User Interface (GUI)-based tools such as Petri Nets [25] for application definition, where the application composition is better visualized. However, this graphical representation is later converted into other forms for further manipulation.

5.1.4. Data requirements. Managing applications in Grids also needs to handle different types of data such as, input data, backend databases, intermediate data products, and output data. Many Bioinformatics applications often have small input and output data but rely on massive backend databases that are queried as part of task execution. On the other hand, some Astronomy applications generate huge output data that are fed into other applications for further processing. Some applications also need the data to be streamed between the tasks for efficient execution.

Thus, the data requirements of an application can be categorized into two types: light and heavy. If an application needs huge amount of data as input or generates massive intermediate or output data products then its data requirement is considered as Heavy. These applications are generally known as data-intensive applications. Whereas, if the application is computation-intensive, it does not need much data to be handled and its data requirement is considered as Light.

5.2. Application scheduling

Effective scheduling is a key concern for the execution of performance-driven Grid applications. Scheduling is a process of finding the efficient mapping of tasks in an application to the suitable resources so that the execution can be completed with the satisfaction of objective functions such as execution time minimization as specified by Grid users. In this section, we discuss application scheduling taxonomy from the perspective of: (a) scheduling architecture; (b) scheduling objective; (c) scheduling decision; and (d) scheduler integration as shown in Figure 8.

5.2.1. Scheduling architecture. The architecture of scheduling infrastructure is very important with regard to scalability, autonomy, and performance of the system [26]. It can be divided into three categories: centralized, hierarchical, and decentralized.

In centralized scheduling architecture [23], scheduling decisions are made by a central controller for all the tasks in an application. The scheduler maintains all information about the applications and keeps track of all available resources in the system. Centralized scheduling organization is simple to implement, easy to deploy and presents few management hassles. However, it is not scalable with respect to the number of tasks and Grid resources.

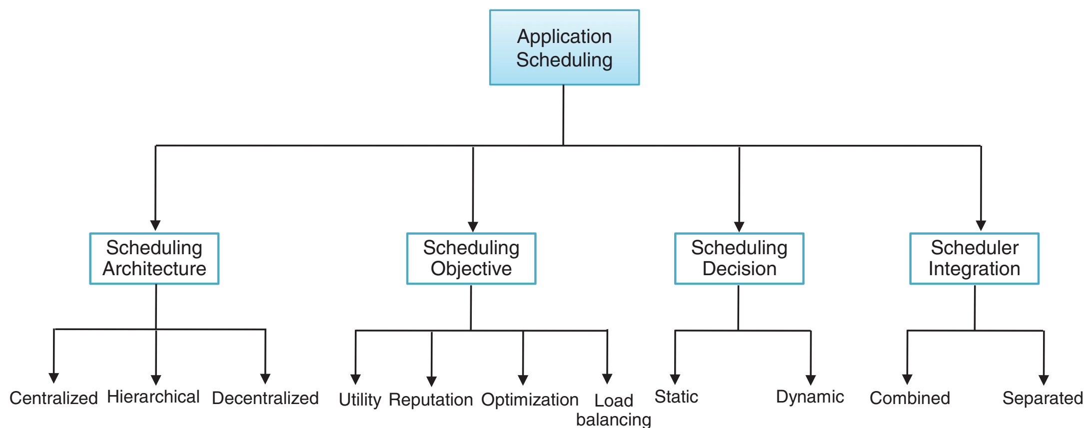

Figure 8. Application scheduling taxonomy.

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

For hierarchical scheduling, there is a central manager and multiple lower-level schedulers. This central manager is responsible for handling the complete execution of an application and assigning the individual tasks of this application to the low-level schedulers. Whereas, each lower-level scheduler is responsible for mapping the individual tasks onto Grid resources. The main advantage of using hierarchical architecture is that different scheduling policies can be deployed at central manager and lower-level schedulers [26]. However, the failure of the central manager results in entire system failure.

In contrast, decentralized scheduler organization [27] negates the limitations of centralized or hierarchical organization with respect to fault-tolerance, scalability, and autonomy (facilitating domain-specific resource allocation policies). This approach scales well since it limits the number of tasks managed by one scheduler. However, this approach raises some challenges in the domain of distributed information management, system-wide coordination, security, and resource provider's policy heterogeneity.

5.2.2. Scheduling objective. The application schedulers generate the mapping of tasks to resources based on some particular objectives. Usually, the schedulers employ an objective function that takes into account the necessary objectives and endeavor to maximize the output. The most commonly used scheduling objectives in a Grid environment are utility, reputation, optimization, and load balancing.

Utility is a measure of relative satisfaction. In a utility-driven approach, the users or service consumers prefer to execute their applications within certain budget and deadline, whereas the resource providers tend to maximize their profits. Reputation refers to the performance of computing resources in terms of successful task execution and trustworthiness. Optimization is related to the improvement of performance with regard to application completion time or resource utilization. Load balancing is also a measure of performance, where the workload on the resources is distributed in such a way so that any specific resource is not overloaded.

5.2.3. Scheduling decision. An application scheduler uses a specific scheduling strategy for mapping the tasks in an application to suitable Grid resources in order to satisfy user requirements. However, the majority of these scheduling strategies are static in nature [28]. They produce a good schedule given the current state of Grid resources and do not take into account changes in resource availability.

On the other hand, dynamic scheduling [29] is done on-the-fly considering the current state of the system. It is adaptive in nature and able to generate efficient schedules, which eventually minimizes the application completion time as well as improves the performance of the system.

5.2.4. Scheduler integration. The scheduler component in a Grid system provides the service of generating execution schedules that map the tasks in an application onto distributed computing resources considering their availability and user's requirements. It also keeps track of the status of the tasks being executed in these Grid resources.

The application scheduler can be deployed in a Grid environment as a scheduling or brokering service to be consumed by the Grid users utilizing the principles of service-oriented architecture (SOA). In this case, the scheduler component is deployed separately in a server and the users submit their applications to this service [30], where the individual task scheduling and submission are managed by the scheduler. On the other hand, scheduler can also be combined or integrated into the system at user's side so that the users are not required to connect another service for the scheduling purpose.

5.3. Coordination

The effectiveness of autonomic application management in a distributed computing environment also depends on the level of coordination among the AEs such as application scheduler or resource broker, local resource management system (LRMS) and resource information service. Lack of coordination among these components may result in communication overhead, which eventually degrades the performance of the system. In general, the process of coordination with respect to

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

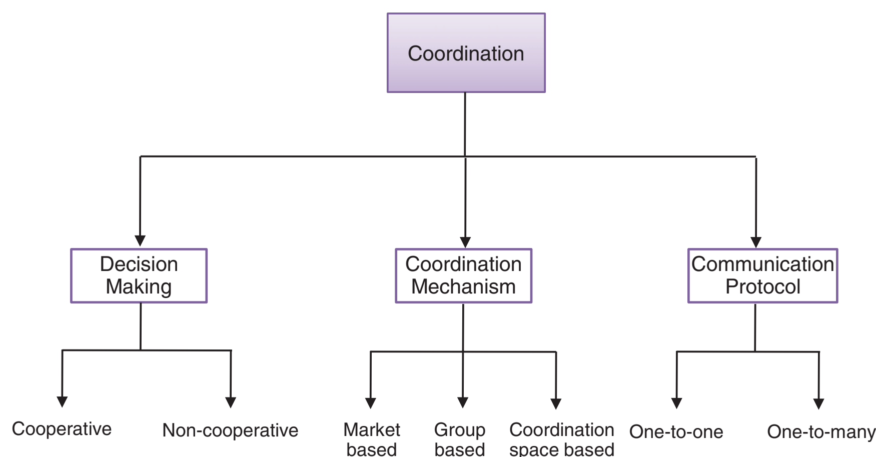

Figure 9. Coordination taxonomy.

application scheduling and resource management in Grids involves dynamic information exchange between various entities in the system. In this section, we discuss the coordination taxonomy from the view of (a) decision making, (b) component integration, and (c) negotiation policy as illustrated in Figure 9.

5.3.1. Decision making. In a distributed computing environment, the autonomic components communicate or interact with each other for the purpose of individual or system-wide decision making (e.g. overlay construction, task scheduling, and load balancing). The process of decision making can be divided into two categories: cooperative and non-cooperative.

In the non-cooperative decision-making scheme, application schedulers perform scheduling-related activities independent of the other schedulers in the system. For example, Condor-G [31] resource brokering system performs non-cooperative scheduling by directly submitting jobs to the condor pools without taking into account their load and utilization status. This approach exacerbates the performance of the system due to load-balancing and utilization problems.

In contrast, the cooperative decision-making approach [32] negotiates resource conditions with the local site managers in the system, if not, with the other application level schedulers. Thus, it is not only able to avoid the potential resource contention problem but also distribute the workload evenly over the entire system.

5.4. Coordination mechanism

Realizing effective coordination among the dynamic and distributed autonomous entities requires robust coordination mechanism and negotiation policies. Three types of coordination mechanisms are well adopted in Grids: market-based coordination, group-based coordination, and coordination space-based coordination.

Market-based mechanism views computational Grids as a virtual marketplace in which economic entities interact with each other through buying and selling computing or storage resources. Typically, this coordination mechanism is used to facilitate efficient resource allocation. One of the common approaches to achieve market-based coordination is to establish agreements between the participating entities through negotiations. Negotiation among all the participants can be done based on a well-known agent coordination mechanism called contract net protocol [33], where the resource provider works as a manager that exports its local resources to the outside contractors or resource brokers and is responsible for decision regarding admission control based on negotiated SLA.

In the collaborative Grid environment, resource sharing is often coordinated by forming groups (e.g. Virtual Organization (VO) [34]) of participating entities with similar interests. In Grids, VO refers to a dynamic set of individuals and institutions defined around a set of resource-sharing rules

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

and conditions. The users and resource providers in a VO share some commonality among them, including common concerns, requirements, and goals. However, the VOs may vary in size, scope, duration, sociology, and structure. Thus, inter-VO resource sharing in Grids is achieved through the establishment of SLA among the participating VOs.

Decentralized coordination space [35] provides a global virtual shared space for the AEs in Grids. This space can be concurrently and associatively accessed by all participants in the system, and the access is independent of the actual physical or topological proximity of the hosts. New generation DHT-based routing algorithms [36, 37] form the basis for organizing the coordination space. The application schedulers post their resource demands by submitting a *Resource Claim* object into the coordination space, whereas resource providers update the resource information by submitting a *Resource Ticket* object. If there is a match between these objects, then the corresponding entities communicate with each other in order to satisfy their interests.

5.5. *Communication protocol*

The interaction among the autonomic components is coordinated by utilizing some particular communication protocols that can be divided into two types: One-to-one and One-to-many. Communication protocols based on One-to-all broadcast are simple but very expensive in terms of number of messages and network bandwidth usage. This overhead can be drastically reduced by adopting One-to-one negotiation among the resource providers and consumers through establishment of SLA.

5.6. *Monitoring*

Monitoring in Grids involves capturing information regarding the environment (e.g. number of active resources, queue size, processor load) that are significant to maintain the self-* properties of the system. This information or monitoring data are utilized by the MAPE-K autonomic loop and the necessary changes are accordingly executed by the autonomic manager through effectors. The sensing components of an AE in Grids require appropriate monitoring data to recognize failure or suboptimal performance of any resource or service. As shown in Figure 10, monitoring can be done in three levels: (a) execution monitoring, (b) status monitoring, and (c) directory service.

5.6.1. Execution monitoring. Once an application is scheduled and the tasks in that application are submitted to corresponding Grid resources for execution, the scheduler needs to periodically monitor the execution status (e.g. queued, started, finished, and failed) of these tasks so that it can efficiently manage the unexpected events such as failure. We identify two types of execution monitoring in Grids: active and passive.

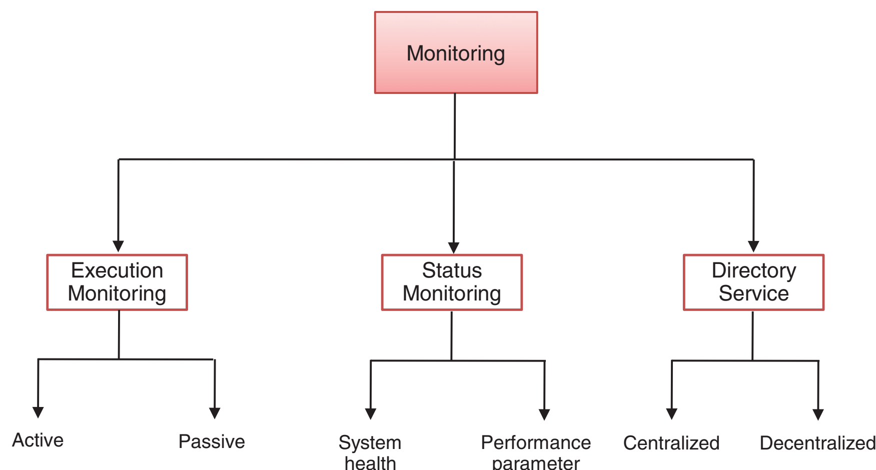

Figure 10. Monitoring taxonomy.

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

The concept of active and passive monitoring of task execution in Grids is derived from the push–pull protocol [38], widely used in the research area of computer network. In active monitoring, information related to task execution is created by engineering the software at some levels, for example, modifying and adding code to the implementation of the application or the operating system to capture function or system calls so that the application itself can report monitoring data periodically (pulling). Moreover, the request for transferring status information is often initiated by the receiver or client in active monitoring strategy. For instance, the application scheduler can send an isAlive probe to the Grid resources currently executing its application for detecting the availability (e.g. online) of these resources at that time.

In contrast, passive monitoring technique captures status information at the resource or server side by the local monitoring service and reports monitoring data to the user or scheduler side periodically (pushing). For example, a resource provider in Grids can periodically inform the status of its system, such as current load to the interested application schedulers according to the requirements specified in the agreement between them.

5.6.2. Status monitoring. The AEs in Grids need to monitor the relevant system properties (i.e. system health) to optimize its operating condition and facilitate efficient decision making. System health data relate to runtime system information, such as memory consumption, CPU utilization, and network usage. The process of collecting system-related information is straightforward and most of the operating systems provide a set of commands (e.g. top, vmstat in Linux) or tools to perform this operation.

In addition, the AE also needs to monitor the performance parameters of its operation such as SLA violation if it incorporates market-based mechanisms to interact with other AEs in the system. To this end, it periodically measures the performance metrics (e.g. service uptime) with regard to the SLA and takes necessary steps to prevent the violation of agreement.

However, the monitoring service often suffers from the dilemma of deciding on how frequently and how much monitoring data should be collected to facilitate efficient decision making. Thus, dynamic and proactive monitoring approaches are essential in order to achieve autonomicity. For example, QMON [39] is an autonomic monitoring service that adapts its monitoring frequency and data volumes for minimizing the overhead of continuous monitoring, while maximizing the utility of the performance data.

5.6.3. Directory service. The directory service provides information about the available resources in the Grid and their status such as, host name, memory size, and processor load. The application schedulers or resource brokers rely on this information for efficiently mapping the tasks in an application to the available resources. Based on the underlying structure, two types of directory services are available in Grids: centralized and decentralized.

In a centralized directory service (e.g. Grid Market Directory [40]), and these monitoring data are stored in a centralized repository. Current studies have shown that [41] the existing centralized model for resource directory services does not scale well as the number of users, brokers, and resource providers increases in the system and vulnerable to single point of failure. Whereas, decentralized information service distributes the process of resource discovery and indexing over the participating Grid sites so that load is balanced and if one site is failed, another site can take over its responsibility autonomously.

5.7. Self- properties*

In this section, we discuss the self-* properties taxonomy from the perspective of four primary properties: (a) self-configuring; (b) self-optimizing; (c) self-healing; and (d) self-protecting (refer to Figure 11). As illustrated in Section 3.1, the evaluation of an autonomic system depends on to what extent it adopts or implements the self-* properties. Further, it is very difficult for a system to fully implement all the self-* properties and in many cases it becomes redundant. Thus most of the autonomic systems focus on some particular properties based on their requirements and goals.

Copyright © 2011 John Wiley & Sons, Ltd.

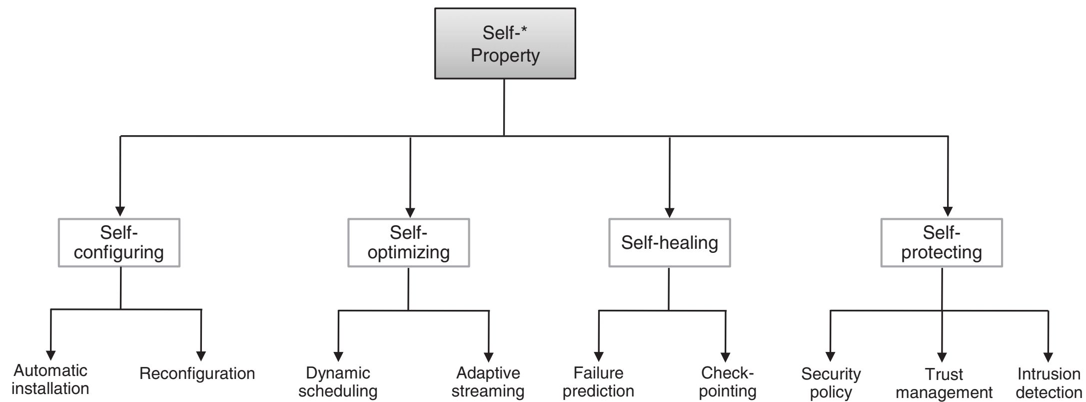

Figure 11. Self-* properties taxonomy.

5.7.1. Self-configuring. Self-configuration refers to the ability to adapt to changes in the system. In regard to autonomic application management, the system can demonstrate the self-configuration property by automatically installing the software components when it detects that some prerequisite components are outdated or missing. This ensures the timely update of the system without human intervention. In addition, the system can also reconfigure itself in the event of dynamic and changing condition.

5.7.2. Self-optimizing. Self-optimization refers to the ability to improve the performance of the system through continuous optimization. In Grids, the AMSs can use dynamic scheduling techniques [42] for mapping application tasks to Grid resources in order to implement the self-optimizing property. The dynamic scheduling approach proactively monitors the status of the Grid resources and schedules tasks according to the current condition of the computational environment by dynamic policies such as rescheduling. Another continuous optimization strategy can be the utilization of adaptive streaming technique for data-intensive applications [43].

5.7.3. Self-healing. Self-healing refers to the ability to discover, diagnose, and recover from faults. Thus, the self-healing property enables a distributed computing system to be fault-tolerant by avoiding or minimizing the effects of execution failures. In a Grid environment, task execution failure can happen for various reasons: (i) the sudden changes in the execution environment configuration, (ii) no availability of required services or software components, (iii) overloaded resource conditions, (iv) system running out of memory, and (v) network failures. In order to efficiently handle these failures an autonomic system can adopt some preemptive policies such as Failure prediction, Check-pointing, and Replication.

Failure prediction techniques [44] are used to predict the availability of the Grid-wide resources continuously over certain period of time. Using these predictions, application schedulers can plan the mapping of tasks to the resources considering their future availability in order to avoid possible task failures. The check-pointing technique [45] transfers the failed tasks transparently to other resources, so that the task can continue its execution from the point of failure. The replication technique [46] executes the same task simultaneously on multiple Grid resources to increase the probability of successful task execution.

5.7.4. Self-protecting. Self-protecting refers to the ability to anticipate and protect against threats or intrusions. This property makes an autonomic system capable of detecting and protecting itself from malicious attacks so as to maintain overall system security and integrity.

Self-protecting application management can be achieved by implementing some proactive policies (i.e. dynamic access control) at both resource and user sides such as, providing accurate warning about potential malicious attack, taking networked resources offline if any anomaly is

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

detected, and shutting down the system if any hazardous event occurs that can possibly damage the system.

One technique for enabling self-protection is to utilize distributed trust management system. A relevant distributed trust mechanism is PeerReview [47]. These distributed trust management systems determine malicious participants through behavioral auditing. An auditor node A checks if it agrees with the past actions of an auditee node B. In case of disagreement, A broadcasts an accusation of B. Interested third-party nodes verify evidence, and take punitive action against the auditor or the auditee.

Another approach to protect the system from malicious attacks is to leverage intrusion detection techniques [48], where the system performs online monitoring and analyzes the attacks/intrusions. Then a model is devised and trained using the past data that is used to successfully and efficiently detect future attacks.

5.8. System characteristics

Owing to its inherent nature, a distributed system possesses some characteristics such as decentralization, heterogeneity, complexity, and reliability. These characteristics not only enforce challenges for designing the system but also make the system useful to the users. As indicated in Figure 12, there are three characteristics of a distributed system, which are related to autonomic application management: (a) complexity, (b)scalability, and (c) volatility. The more a system becomes complex or volatile, the more it needs to incorporate AC principles in order to avoid performance degradation and user satisfaction. Whereas, increasing the scalability of a system facilitates itself to adopt autonomic features gracefully.

5.8.1. Complexity. According to the definition of Buyya *et al.* [49], a Grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous resources dynamically at runtime depending on their availability, capability, performance, cost, and users' QoS requirements. The resources are heterogeneous and fault-prone as well as may be administered by different organizations. Thus Grid systems are complex by their characteristics.

However, as the complexity of a system increases, it becomes brittle and unmanageable. In that case, the system needs to be more autonomous so that it can handle the consequences of complexity without human intervention. For instance, the complexity of a decentralized Grid system is much higher than a centralized Grid system because of the interaction and coordination of the large number of decentralized components.

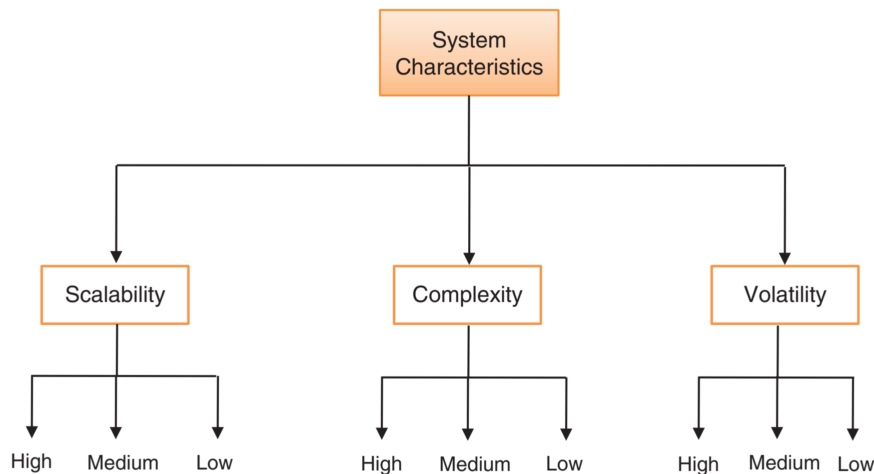

Figure 12. System characteristics taxonomy.

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

5.8.2. Scalability. Scalability of a distributed system is considered as the ability for the system to easily expand or contract its resource pool to accommodate heavier or lighter loads. In other words, scalability is the ease with which a system can be modified, added or removed in order to accommodate varying workload.

If a system is highly scalable, its performance should not dramatically deteriorate as the system size increases. In general, decentralized or P2P Grid systems [50] are scalable in nature, whereas the centralized Grid systems [51] are least scalable as there exists a single point of control.

5.8.3. Volatility. Volatile refers to changing or changeable. Thus, volatility of a distributed system can be defined as the likelihood that the status of the system or its components to be altered due to the heterogeneous and dynamic behavior of the environment such as, configuration change, resource failure and load variation. If the condition of the system changes frequently over a short period of time, it has high volatility. If the system status almost never changes, it has low volatility. In general, Grid systems are volatile in nature due to the underlying characteristics.

6. SURVEY OF GRID SYSTEMS

This section provides a detailed survey of selected existing Grid systems and mapping of the taxonomy proposed in the previous section onto these systems. Table III shows the summary of selected Grid workflow management projects. A comparison of various Grid systems and their categorization based on the taxonomy is shown in Tables IV–IX.

6.1. Aneka federation

Aneka Federation system [50] logically connects topologically and administratively distributed Aneka Enterprise Grids as part of a single cooperative system. It uses a Distributed Hash Table (DHT) such as Pastry [37], Chord [36]-based self-configuring P2P network model for discovering

Table III. Summary of Grid projects.

Name	Organization	Status/Availability	Application focus
Aneka Federation	The University of Melbourne, Australia http://www.gridbus.org	Free Evaluation version	Compute-intensive Bag-of-task
Askalon	University of Innsbruck, Austria http://dps.uibk.ac.at/askalon	Under Askalon Software License	Performance-oriented scientific Workflow
AutoMate	Rutgers University, USA http://www.caip.rutgers.edu/TASSL/Projects/AutoMate/	N/A	Data-intensive Bag-of-task
Condor-G	University of Wisconsin, USA http://www.cs.wisc.edu/condor/condorg/	Source code under Apache License	Compute-intensive Bag-of-task
GWMS	The University of Melbourne, Australia http://www.gridbus.org	Open source under GPL	Computational and data-intensive Workflow
Nimrod-G	Monash University, Australia http://messagelab.monash.edu.au/NimrodG	Source code under DSTC license	Computational and data-intensive Bag-of-task
Pegasus	University of Southern California, USA http://pegasus.isi.edu	Open source under Atlassian Confluence	Data-intensive Workflow
Taverna	Collaboration between several European institutes and industries http://taverna.sourceforge.net/	Source code under LGPL license	Bioinformatics Workflow
Triana	Cardiff University, UK http://www.trianacode.org/	Source code under Apache License	Compute-intensive Workflow

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

Table IV. Application composition taxonomy.

Project	Application type	Application domain	Application definition	Data requirement
Aneka Federation	Bag-of-task	Business/Scientific	XML-based custom	Light
Askalon	Workflow	Scientific	AGWL-based custom	Light/Heavy
AutoMate	Bag-of-task/Workflow	Business/Scientific	XML-based custom	Heavy
Condor-G	Bag-of-task/MPI	Scientific	ClassAd-based custom	Light
GWMS	Workflow	Scientific	xWFL-based custom	Heavy
Nimrod-G	Bag-of-task/MPI	Business/Scientific	DPML-based custom	Heavy
Pegasus	Workflow	Scientific	VDL-based custom	Heavy
Taverna	Workflow	Scientific	Scufl-based custom	Heavy
Triana	Workflow	Scientific	Tool-based	Light

Table V. Application scheduling taxonomy.

Project	Scheduling architecture	Scheduling objective	Scheduling decision	Scheduler integration
Aneka Federation	Decentralized	Load balancing	Dynamic	Combined
Askalon	Centralized	Utility/Optimization	Static/Dynamic	Separated/Combined
AutoMate	Decentralized	Load balancing	Dynamic	Combined
Condor-G	Centralized	Load balancing	Dynamic	Combined
GWMS	Centralized	Utility/Optimization	Dynamic	Separated
Nimrod-G	Centralized	Utility/Optimization	Dynamic	Separated/Combined
Pegasus	Centralized	Optimization	Static/Dynamic	Separated
Taverna	Centralized	Optimization	Dynamic	Separated/Combined
Triana	Centralized/ Decentralized	Optimization	Dynamic	Separated/Combined

Table VI. Coordination taxonomy.

Project	Decision making	Coordination Mechanism	Communication Protocol
Aneka Federation	Cooperative	Coordination space based	One-to-many
Askalon	Non-cooperative	Market/Group based	One-to-one
AutoMate	Cooperative	Coordination space based	One-to-many
Condor-G	Non-cooperative	Group based	One-to-one
GWMS	Non-cooperative	Market/Group based	One-to-one
Nimrod-G	Non-cooperative	Market/Group based	One-to-one
Pegasus	Non-cooperative	Group based	One-to-one
Taverna	Non-cooperative	Group based	One-to-one
Triana	Cooperative	Group based	One-to-many (all)

and coordinating the provisioning of distributed resources in Aneka Grids. It also employs a novel resource provisioning technique that assigns the best possible resource sets for the execution of applications, based on their current utilization and availability in the system.

Copyright © 2011 John Wiley & Sons, Ltd.

Table VII. Monitoring taxonomy.

Project	Execution monitoring	Status monitoring	Directory Service
Aneka Federation	Passive	System health	Decentralized
Askalon	Passive	System health	Centralized
AutoMate	Passive	System health/Performance parameter	Decentralized
Condor-G	Active	System health	Centralized
GWMS	Passive	System health/Performance parameter	Centralized
Nimrod-G	Passive	System health/Performance parameter	Centralized
Pegasus	Active	System health	Centralized
Taverna	Passive	System health	Centralized
Triana	Passive	System health	Decentralized

Table VIII. Self-* properties taxonomy.

Project	Self-configuring	Self-optimizing	Self-healing	Self-protecting
Aneka Federation	Reconfiguration	Dynamic rescheduling	Failure detection	NA
Askalon	NA	Dynamic rescheduling/ Performance prediction	Failure detection/ Check-pointing	Authentication detection
AutoMate	Reconfiguration	Adaptive streaming	NA	Security policy
Condor-G	NA	Dynamic rescheduling	Failure detection	NA
GWMS	NA	Dynamic rescheduling	Failure detection	NA
Nimrod-G	NA	Dynamic rescheduling	Failure detection	NA
Pegasus	NA	Dynamic rescheduling	Failure detection/ Check-pointing	Authentication detection
Taverna	NA	Dynamic rescheduling	Failure detection	NA
Triana	Reconfiguration	Dynamic rescheduling	Failure detection/ Check-pointing	NA

Table IX. System characteristics taxonomy.

Project	Scalability	Complexity	Volatility
Aneka Federation	High	High	High
Askalon	Medium	Medium	Medium
AutoMate	High	High	High
Condor-G	Low	Low	Medium
GWMS	Low	Medium	Medium
Nimrod-G	Low	Medium	Medium
Pegasus	Low	Medium	Medium
Taverna	Low	Medium	Medium
Triana	High	High	High

The application scheduling and resource discovery in Aneka-Federation is facilitated by a specialized Grid Resource Management System, known as Aneka Coordinator (AC). AC is composed of three software entities: Grid Resource Manager (GRM), LRMS, and Grid Peer. The GRM component of AC exports a Grid site to the federation and is responsible for coordinating federation-wide application scheduling and resource allocation. GRM is also responsible for scheduling locally submitted jobs in the federation using LRMS.

Grid peer implements a DHT-based P2P overlay for enabling decentralized and distributed resource discovery supporting resources status lookups and updates across the federation. It also

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

enables decentralized inter-AC collaboration for optimizing load-balancing and distributed resource provisioning. The employment of DHT improves system scalability by enabling the ability to perform deterministic discovery of resources and produce controllable number of messages (by using selective broadcast approach) in comparison with using other One-to-All broadcast techniques such as JXTA [52].

Distributed trust mechanism is utilized in Aneka Federation to ensure secured resource management across the federation. Furthermore, the Aneka Container component of AC provides the base infrastructure that consists of services for persistence and security (authorization, authentication, and auditing).

6.2. Askalon

Askalon [53] is a Grid application development and execution environment. The goal of Askalon is to simplify the development and optimization of mostly scientific workflow applications that can harness the power of computational Grids (i.e. Austrian Grid). Askalon comprises four tools (Scalea, Zenturio, Aksum, PerformanceProphet), coherently integrated into an SOA. Scalea is a performance measurement and analysis tool for parallel and distributed high-performance applications. Zenturio is a general-purpose experiment management tool with the support for multi-experiment performance analysis and parameter studies. Through a special-purpose performance property specification language, Aksum provides semi-automatic high-level performance bottleneck detection. The PerformanceProphet facilitates the users in terms of modeling and predicting the performance of parallel applications at the early stages of development.

Askalon uses the XML-based Abstract Grid Workflow Language (AGWL) [24] for composing workflow applications. The Scheduler service processes the workflow specification described in AGWL, converts it into an executable form, and maps it onto the available Grid resources. Resource Manager is utilized to retrieve the current status and availability of Grid resources. Furthermore, the Enactment Engine coordinates the execution of workflow tasks according to the control flows and data dependencies specified by the application developers.

Askalon employs a hybrid approach for scheduling workflow applications on the Grid through dynamic monitoring and steering, combined with a static optimization. Static optimization maps entire workflow onto the Grid resources using Genetic Algorithm based on user-defined QoS parameters. A dynamic scheduling algorithm then takes into consideration the dynamic nature of the Grid resources, such as machine crashes or external CPU and network load. Askalon also employs the self-healing mechanism through checkpointing and migration techniques that support reliable workflow execution in the presence of resource failures as well as when the Enactment Engine itself crashes.

In order to establish authentication mechanism across Askalon user portals and Grid services, Grid Security Infrastructure [54] has been employed based on single sign-on, credential delegation, and web services security (through XML digital signature and XML encryption).

6.3. AutoMate

The main objective of AutoMate [10] is to develop conceptual models and implementation architectures that can enable the development and execution of self-managing Grid applications. The major components of AutoMate are: Accord programming framework, Rudder coordination middleware, Meteor content-based middleware, and Sesame access control engine. In AutoMate, application composition plans are generated by Accord, element discovery is performed by Meteor, and plan execution is achieved by Rudder. AutoMate portals provide users with secure, pervasive and collaborative access to different components and entities.

The Accord programming framework extends existing distributed programming models and frameworks to address the definition, execution, and runtime management of AEs. In particular, it extends the entities and composition rules defined by the underlying programming model to enable computational and composition/interaction behaviors to be defined at runtime using high-level rules. In Accord, composition plans are generated using the Accord Composition Engine and are expressed in XML.

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

The Rudder coordination middleware provides the core capabilities for supporting autonomic composition, adaptation and optimizations. It builds upon two concepts: Context-aware agent framework and Decentralized tuple space. A context-aware agent is a processing unit that performs tasks to automate the control and coordination of the AEs. The decentralized tuple space scalably and reliably supports the distributed agent-based system coordination.

The Meteor content-based middleware provides support for content-based routing, decentralized information discovery, and messaging service through Squid and Pawn. Squid is a DHT [55]-based P2P system that enables efficient and scalable information discovery, while supporting complex queries. Pawn is a P2P messaging substrate that builds on JXTA [52] to support P2P interactions in the Grid. Pawn provides a stateful and guaranteed messaging to enable key application-level interactions such as synchronous/asynchronous communication, dynamic data injection, and remote procedure calls.

In AutoMate, secure interaction among the participating entities is managed by AutoMate Access Control Engine, Sesame. It is composed of access control agents and provides dynamic role-based access control to users, applications, services, and resources.

6.4. Condor-G

Condor-G [31] is the combination of technologies from the Condor project and the Globus project [56]. It combines the inter-domain resource management protocols of the Globus Toolkit and the intra-domain resource and job management mechanisms of Condor. In particular, Condor-G leverages security and resource access in multi-domain environments, as supported by the Globus Toolkit as well as management of computation and harnessing of resources within a single administrative domain, embodied by the Condor system. Its flexible and intuitive commands are appropriate for use directly by end-users, or for interfacing with higher-level task brokers and web portals.

The computation management service of Condor-G is called Condor-G agent. It allows users to treat the Grid as an entirely local resource, with an API and command line tools that facilitate them to perform several job management operations: (1) submit jobs by indicating an executable name, input/output files, and arguments; (2) query a job's status or cancel the job; (3) be informed of job termination/problems via callbacks or asynchronous mechanisms such as e-mail; (4) obtain access to detailed log that provides a complete history of the jobs' execution.

Condor-G comprises a powerful, full-featured task broker/scheduler that can manage thousands of jobs destined to run at distributed sites. It supports job scheduling, monitoring, policy enforcement, fault tolerance, credential management, and handles complex job-interdependencies. Specifically, the job-interdependencies are handled by the associated meta-scheduler, Directed Acyclic Graph Manager (DAGMan) [45]. While Condor-G aims to discover available machines for the execution of jobs, DAGMan handles the dependencies between the jobs. The resource brokering is done through a matchmaking algorithm.

The Condor-G scheduler responds to a user request of submitting jobs to be run on Grids by creating a new Condor-G GridManager daemon. One GridManager process handles all jobs for a single user and terminates once all jobs are completed. The job submission request of each GridManager results in the creation of one Globus JobManager [56] daemon. The GridManager detects remote failures by periodically probing the JobManagers of all the jobs it manages and resubmits the failed jobs once detected.

6.5. GWMS

Gridbus Workflow Management System (GWMS) [23] facilitates users to execute their workflow applications on Grids. The two main components of GWMS are workflow portal and workflow engine. The primary user interface for the users to access GWMS is a web portal that comprises an editor and a monitor component. The workflow editor provides a GUI and allows users to create new and modify existing workflows utilizing the drag and drop facilities. Workflow monitor provides a GUI for viewing the status of each task in the workflow. Users can also view the site of execution for each task, the number of tasks being executed, and the failure history of each task.

Copyright © 2011 John Wiley & Sons, Ltd.

Workflow engine is designed to support an XML-based WorkFlow Language (xWFL). This facilitates user-level planning at the submission time. The workflow language parser converts workflow description from XML format to Tasks, Parameters, and Data Constraints (workflow dependency), which are accessed by workflow scheduler. The resource discovery component of the engine sends query to Grid Information Services such as Globus MDS [57], directory service, and replica catalogues to locate suitable resources for the execution of the tasks in the workflow. It also uses Gridbus Broker [30] for dispatching and managing task executions on different Grid sites comprising various middlewares. Execution of workflow tasks on different Grid middlewares is achieved by creating specific dispatchers for corresponding middleware.

Workflow engine also employs a just-in-time scheduling system using tuple space, where every task has its own scheduler called Task Manager (TM), which implements a scheduling algorithm and handles the processing of tasks. The TMs are controlled by a Workflow Coordinator. Although these TMs work in a distributed fashion, they communicate with each other through the tuple space that is designed based on a client-server-based centralized technology. However, the just-in-time scheduling system allows resource allocation decision to be made at the time of task execution and hence adapt to the changing Grid environments. Task failures are handled in GWMS by resubmitting the failed tasks to resources that do not have failure history for these tasks.

6.6. Nimrod-G

Nimrod/G [58] is a widely adopted Grid middleware environment for building and managing large computational experiments over distributed resources. It uses the Globus [56] middleware services for dynamic resource discovery and job dispatching in computational Grids. The main components of Nimrod/G are: Client or User Station, Parametric Engine, Scheduler, Dispatcher, and Job-Wrapper. In addition, it provides a web-based interface that allows users to create and manage experiments without installing Nimrod/G client locally.

Client or User Station acts as a user-interface for controlling and supervising an experiment under consideration. The user can vary parameters related to execution time and cost, which influence the scheduling decision while selecting resources. It also serves as a monitoring console for the users and lists status of all jobs.

The Parametric Engine is the core component of Nimrod/G and acts as a persistent job control agent that handles the whole experiment. It is responsible for parameterization of the experiment, creation of jobs, maintenance of job status, and interaction with Nimrod scheduler and clients. The engine takes the experiment plan, described using a Declarative Parametric Modeling Language (DPML) as input and manages the experiment under the direction of scheduler.

The Scheduler is responsible for resource discovery, resource selection, and job assignment. The resource discovery process interacts with a Grid information service directory (MDS in Globus), identifies the list of authorized machines, and keeps track of resource status information. As Nimrod/G incorporates computational economy-based job scheduling approach, the resource selection process selects the resources that meet the execution completion deadline set by the user as well as minimizes the cost of computation.

The Dispatcher primarily initiates the execution of tasks in a job on the selected resources according to scheduler's instruction. The Job-wrapper is responsible for staging the application data, starting execution of the tasks on assigned resources, and sending results back to the Parametric Engine through Dispatcher. The architecture of Nimrod/G is extensible enough to support job execution in several Grid middleware services, such as Legion [59] and Condor [60] by implementing specific job dispatcher for the corresponding middleware.

6.7. Pegasus

Pegasus Workflow Management System (PWMS) [61] has been developed as part of the GriPhyN project [62] that aims to support large-scale data management in physics experiments, such as high-energy physics and astronomy. It can map and execute complex scientific workflows on the Grid. PWMS is composed of two major components: Pegasus workflow mapping engine and DAGMan workflow executor for Condor.

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

Pegasus workflow mapping engine receives an abstract workflow description expressed in Chimera's [63] Virtual Data Language (VDL) and generates an optimized concrete workflow by mapping workflow tasks to a set of available Grid resources. The abstract workflow describes the tasks and data in terms of their logical names and indicates their dependencies in the form of DAG, whereas concrete workflow specifies the location of the data and the task execution platforms. Pegasus uses the centralized meta-scheduler, DAGMan [45] as enactment engine with the enhancement of data derivation techniques that simplify the workflow at runtime based on data availability. Thus, combined with DAGMan, Pegasus is able to map and execute workflows on a variety of platforms, such as Condor pools, Cluster managed by LSF or PBS, TeraGrid hosts and individual hosts.

In order to locate the replicas of the required data and find the location of logical application components, Pegasus uses Replica Location Service (RLS) and Transformation Catalog (TC), respectively. It also queries Globus Monitoring and Discovery Service (MDS) to find out available resources and their characteristics.

Pegasus uses two methods for resource selection: random allocation and performance prediction. In the latter approach, Pegasus interacts with Prophesy [64] that is used to predict the best site to execute an application component by using performance historical data. Recently, it has adopted the strategy of dynamically adjusting resource allocation decisions in response to feedback on the performance of workflow execution. This adaptive strategy is structured around the MAPE functional decomposition, which partitions the adaptive functionalities into four areas: Monitoring, Analysis, Planning, and Execution. In case of job failure, Pegasus incorporates Retry policy and generates a rescue DAG by DAGMan, which is modified and resubmitted at a later time.

6.8. Taverna

Taverna [65] is a workflow management tool of the myGrid project [66], which aims to exploit Grid technology to develop high-level middleware for supporting data-intensive *in silico* bioinformatics experiments using distributed resources. The tool includes a workbench application, called Scufl Workbench that provides a GUI for the composition of workflows and an enactment engine, called Freefluo enactor that facilitates transferring intermediate data and invoking web services.

In Taverna, a workflow is considered to be a graph of processors, each of which transforms a set of data inputs into a set of data outputs. These workflows are written in a new language, called the Simple Conceptual Unified Flow Language (SCUFL), where each processor within a workflow represents one atomic task.

The Scufl workbench enables bioinformaticians to compose workflows without having to learn SCUFL. It acts as a container for a number of user interface components and provides a user-friendly multi-window environment for users to manipulate workflows, validate and select available resources as well as execute and monitor these workflows. Scufl language parser is used to parse Scufl workflow definitions into a form that is enacted by the Freefluo enactor.

Workflows are executed in the Scufl workbench using the enactor launch panel. This panel allows inputs to be specified for the workflow and launches a local instance of the Freefluo enactment engine. Freefluo is a Java-based workflow orchestration tool. It supports invoking different types of services, such as WSDL-based [20] single operation web services, Soaplab bio-services, Talisman, and local applications. The enactment status panel of Taverna shows the current progress of a workflow invocation and allows users to browse the intermediate and final results.

Fault tolerance in Taverna is achieved by setting configuration (e.g. number of retries, time delay, alternative processor) for each processor in the workflow. It also allows users to specify the critical level for faults on each processor. If a processor is set as Critical, when all retries and alternatives are failed, entire workflow execution is terminated; otherwise, the execution of workflow is continued, but children nodes of the failed processor are never invoked.

6.9. Triana

Triana [67] is a workflow composition and management environment that consists of an intuitive GUI for application composition and an underlying subsystem, which allows integration of Triana

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

with multiple Grid services and interfaces. The GUI consists of two main components: Tool browser and Work surface. Tool browser employs a conventional file browser interface and Work surface is used to graphically connect the tools to form a dataflow diagram. Thus, users create applications by dragging the desired tools (or services) from the Tool browser onto the Work surface, and then wiring them together to create a workflow or dataflow for specific behavior.

The underlying subsystem consists of a collection of interfaces, consisting of various middlewares and services, including the Grid Application Toolkit (GAT) that integrates Triana into the Grid. GAT defines a high-level API for access to core Grid services using JXTA [52], web services, and OGSA (Open Grid Services Architecture). Triana supports two types of application components for distributed execution: Grid-oriented and Service-oriented. Grid-oriented components refer to applications that are executed on the Grid using GRM, such as Grid Resource Management System (GRMS). Service-oriented components are remote applications that are invoked through network interfaces, such as Web services, JXTA services.

An important feature of Triana is that it enables users to distribute sections of a workflow to remote machines for execution. The distribution of workflow requires the existence of Triana launcher services running on the remote machines. A launcher service provides the contact point for Triana on the remote machine and facilitates the creation of actual Triana services executing workflow subsections. Moreover, Triana connects input and output pipes to nodes of the remote service for enabling the data to be passed from the local workflow to the remote service, and the results to be passed back to the user.

The distribution mechanism is facilitated by the GAT interface that is not bound to any specific middleware. Currently, Triana workflow environment supports two types of GAT bindings, one for JXTA, which is a set of protocols for decentralized P2P applications and another for P2PS, a simple socket-based P2P toolkit.

7. FUTURE RESEARCH DIRECTIONS

Today, Small and Medium Business Enterprises (SMEs), universities, and governments face accelerated business change, more intense domestic and global competition and increased IT demands. They try to meet new demands through rapid implementation of innovative and inclusive business models while at the same time lowering IT barriers to innovation and change. These demands [68] call for a more dynamic computing model that supports rapid innovation for services and their delivery. Cloud computing, [49, 69] which can be an important component of such a model, is a recent advance wherein IT-related functionalities (e.g. applications or storage) are provided 'as a service' to end-users under a usage-based payment model. In a Cloud computing model, end-users (SMEs, governments, universities) can leverage virtualized services probably on the fly based on fluctuating requirements and, in doing so, they avoid worry about infrastructure details such as where these resources are hosted or how they are managed. The new computing environment, buoyed by recent advances in the above areas, has resulted in hybrid systems comprises virtualized resources (computing servers, storage), applications, usage-based payment models, and networked devices. The benefit of such an environment is efficiency and flexibility, through creation of a more dynamic computing environment, where the supported functionalities are no longer fixed or locked to the underlying infrastructure. This offers tremendous automation opportunities in a variety of computing domains including, but not limited to, e-Government, e-Research, web hosting, social networking, and e-Business.

Therefore, existing Grid domains (private domain and VO-specific computing environments) and Clouds (Amazon EC2 [70], Microsoft Azure [71], GoGrid [72]) can be pooled together to form a hybrid computing environment of resource pools (nodes, services, virtual machines, storage). In a hybrid computing environment: (i) system can grow or shrink based on demand and operating environment (power failure, heat dissipation, natural disasters); (ii) the peak-load handling capacity of every Grid computing domain is enhanced without having the need to maintain or administer any additional hardware or software infrastructure; and (iii) the ability of computing domain as regards

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

reliable service delivery is augmented due to availability of multiple redundant resource pools that can efficiently tackle disaster conditions and ensure continuity of crucial business and scientific applications. The major research challenges that need to be solved for supporting aforementioned hybrid computing environments include [73]:

- Application Service Behavior Prediction: It is critical that the AMS is able to predict the demands and behaviors of the application services to be deployed on the resources that belong to hybrid environments, so that it can intelligently undertake decisions related to dynamic scaling or de-scaling of applications and resources.
- Flexible Mapping of Applications to Resources: With increased operating costs and energy requirements of hybrid environments [74], it becomes critical to maximize their efficiency, cost-effectiveness, and utilization. The process of mapping services to resources is a complex undertaking, as it requires the system to compute the best software and hardware configuration (system size and mix of resources) to ensure that QoS targets of services are achieved, while maximizing system efficiency and utilization.
- Combinatorial Optimization Techniques: Deployment plan for application services over hybrid environments is combinatorial optimization problem that searches the optimal combinations of resources, services, and their deployment plans. Unlike many existing multi-objective optimization solutions, the optimization models that ultimately aim to optimize both resource-centric (utilization, availability, reliability, incentive) and user-centric (response time, budget spent, fairness) QoS targets need to be developed.
- Integration and Interoperability: For many organizations, there is a large amount of IT assets in-house, in the form of line of business applications that are unlikely to ever be migrated to the Cloud. Further, there is huge amount of sensitive data in an enterprise, which is unlikely to migrate to the Cloud due to privacy and security issues. As a result, there is a need to look into issues related to integration and interoperability between the software on premises and the application services in the Cloud.
- Scalable Monitoring of System Components: Although the components that contribute to a hybrid computing environment may be distributed, existing techniques usually employ centralized approaches to overall system monitoring and management. We claim that centralized approaches are not an appropriate solution for this purpose, due to concerns of scalability, performance, and reliability arising from the management of multiple service queues and the expected large volume of service requests. Therefore, we advocate architecting next-generation service monitoring and management services based on decentralized messaging [75] (such as P2P) and indexing models.

8. CONCLUSION

This paper presents an overview and state-of-the-art of Autonomic Application Management (AAM) in Grid computing environment. After analyzing the AAM landscape, we categorize the process of facilitating AAM according to various aspects of application management and propose a comprehensive taxonomy based on six different perspectives: (i) application composition, (ii) application scheduling, (iii) coordination, (iv) monitoring, (v) self-* property, and (vi) system characteristics. Further, we develop taxonomies for each of these perspectives to classify the common trends, solutions, and techniques in application management. Hereby, we provide pointers to related research work in this context. Next, a survey is also conducted, where the taxonomy is mapped to the selected Grid systems mostly focused on scientific workflow management. The survey helps to analyze the gap between what autonomic application management policies and methodologies are already available in existing Grid systems and what are still required to be addressed so that some outstanding research issues can be identified.

From the taxonomy and survey, we identify that most of the existing Grid workflow management systems are centralized and do not support cooperative application scheduling. In addition, as these Grid systems are highly complex and volatile, most of them incorporate self-optimizing and

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

self-healing properties of an ACS. However, in order to cope up with the increasing-scale complexity and volatility of Grid environment, these systems are also required to address the self-configuring and self-protecting policies to some extent.

REFERENCES

1. Parashar M, Hariri S. *Autonomic Computing*: *An Overview* (*Lecture Notes in Computer Science*, vol. 3566). Springer: Berlin, 2005; 247–259.
2. Available from: http://www.darpa.mil/sto/strategic/suosas.html [17 November 2010].
3. Huebscher MC, McCann JA. A survey of autonomic computing—Degrees, models, and applications. *ACM Computing Surveys* 2008; **40**(3):1–28.
4. Muscettolay N, Nayakz P, Pellz B, Williams B. Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence* 1998; **103**(1–2):5–47.
5. Chen J, Yang Y. Temporal dependency based checkpoint selection for dynamic verification of fixed-time constraints in grid workflow systems. *Proceedings of the 30th International Conference on Software Engineering*, *ICSE '08*. ACM: New York, NY, U.S.A., 2008; 141–150.
6. Ganek A, Corbi T. The drawing of the autonomic computing era. *IBM Systems Journal*, *Special Issue on Autonomic Computing* 2003; **24**(1):5–18.
7. Horn P. Autonomic computing: Ibm's perspective on the state of information technology. *Technical Report*, IBM Corporation, October, 2001.
8. An architectural blueprint for autonomic computing. *Technical Report*, IBM Corporation, 2003.
9. Parashar M, Hariri S. Autonomic grid computing. *Proceedings of International Conference on Autonomic Computing*, U.S.A., May 2005.
10. Parashar M, Liu H, Li Z, Matossian V, Schmidt C, Zhang G, Hariri S. Automate enabling autonomic applications on the grid. *Cluster Computing*: *The Journal of Networks*, *Software Tools*, *and Applications*, *Special Issue on Autonomic Computing* 2006; **9**(1):161–174.
11. Kim H, Parashar M, Foran DJ, Yang L. Investigating the use of autonomic cloudbursts for high-throughput medical image registration. *Proceedings of 10th IEEE/ACM International Conference on Grid Computing*, Alb., Canada, October 2009.
12. Russell S, Norvig P. *Artificial Intelligence*: *A Modern Approach*. Prentice-Hall: Englewood Cliffs, NJ, U.S.A., 2003.
13. A technical view of autonomic computing. Software Group, IBM Corporation, U.S.A., 2002.
14. Cirne W, Brasileiro F, Sauve J, Andrade N, Paranhos D, Santos-Neto E, Medeiros R. Grid computing for bag of tasks applications. *Proceedings of the 3rd IFIP Conference on E-Commerce*, *E-Business and E-Government*, Brazil, September 2003.
15. Nascimento P, Sena C, da Silva J, Vianna D, Boeres C, Rebello V. Managing the execution of large scale mpi applications on computational grids. *Proceedings of the 17th International Symposium on Computer Architecture on High Performance Computing* (*SBAC-PAD*), Brazil, October 2005.
16. Ramakrishnan L, Gannon D. A survey of distributed workflow characteristics and resource requirements.
17. Laure E, Jones B. Enabling grids for e-science: The egee project. *Technical Report EGEE-PUB-2009-001*, CERN, 2009.
18. Better business using grid solutions, bingrid.
19. Extensible markup language (xml) 1.0 (third edition).
20. Web services description language (wsdl) version 1.2.
21. Juric MB, Mathew B, Sarang P. *Business Process Execution Language for Web Services*. Packt Publishing: U.K., 2004.
22. World wide web consortium (w3c).
23. Yu J, Buyya R. *Gridbus Workflow Enactment Engine*, *Grid Computing*: *Infrastructure*, *Service*, *and Applications*, Wang L, Jie W, Chen J (eds). CRC Press: U.S.A., 2009.
24. Fahringer T, Qin J, Hainzer S. Specification of grid workflow applications with agwl: An abstract grid workflow language. *Proceedings of International Symposium on Cluster Computing and the Grid* (*CCGrid '05*), Cardiff, U.K., May 2005.
25. Peterson JL. Petri nets. *ACM Computing Surveys* 1977; **9**(3):223–252.
26. Hamscher V, Schwiegelshohn U, Streit A, Yahyapour R. Evaluation of job-scheduling strategies for grid computing. *Proceedings of 1st IEEE/ACM International Workshop on Grid Computing* (*Grid'00*), Berlin, 2000.
27. Ranjan R, Rahman M, Buyya R. A decentralized and cooperative workflow scheduling algorithm. *Eighth IEEE International Symposium on Cluster Computing and the Grid* (*CCGrid'08*), France, May 2008.
28. Topcuoglu H, Hariri S, Wu MY. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems* 2002; **13**(3):260–274.
29. Rahman M, Venugopal S, Buyya R. A dynamic critical path algorithm for scheduling scientific workflow applications on global grids. *Third IEEE International Conference on e-Science and Grid Computing* (*eScience'07*), Bangalore, India, December 2007.
30. Venugopal S, Buyya R, Winton L. A grid service broker for scheduling e-science applications on global data grids. *Concurrency and Computation*: *Practice and Experience* 2006; **18**((6):685–699.

Copyright © 2011 John Wiley & Sons, Ltd.

31. Frey J, Tannenbaum T, Livny M, Foster I, Tuecke S. Condor-G: a computation management agent for multi-institutional grids *Tenth IEEE International Symposium on High Performance Distributed Computing*, U.S.A., June 2001.
32. Rahman M, Ranjan R, Buyya R. Cooperative and decentralized workflow scheduling in global grids. *Future Generation Computer Systems* (*FGCS*). 2010; **26**(5):753–768.
33. Smith RG. The contract net protocol: high-level communication and control in a distributed problem solver. *Distributed Artificial Intelligence*. Morgan Kaufman Publishers Inc.: San Francisco, CA, U.S.A., 1988; 357–366.
34. Foster I, Kesselman C, Tuecke S. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications* 2001; **15**(3):200–222.
35. Li Z, Parashar M. Comet: A scalable coordination space for decentralized distributed environments. *Second International Workshop on Hot Topics in Peer-to-Peer Systems*, San Diego, U.S.A., 2005.
36. Stoica I, Morris R, Karger D, Kaashoek MF, Balakrishnan H. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Conference on Applications*, *Technologies*, *Architectures*, *and Protocols for Computer Communications*, U.S.A., 2001.
37. Rowstron A, Druschel P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms*, Heidelberg, Germany, 2001.
38. Russo A, Cigno RL. Push/pull protocols for streaming in p2p systems. *Proceedings of the 28th IEEE International Conference on Computer Communications Workshops*, Brazil, 2009.
39. Agarwala S, Chen Y, Milojicic DS, Schwan K. Qmon: Qos- and utility-aware monitoring in enterprise systems. *Proceedings of the 3rd IEEE International Conference on Autonomic Computing* (*ICAC'06*), Ireland, 2006.
40. Yu J, Venugopal S, Buyya R. A market-oriented grid directory service for publication and discovery of grid service providers and their services. *The Journal of Supercomputing* 2006; **36**(1):17–31.
41. Zhang X, Freschl JL, Schopf JM. A performance study of monitoring and information services for distributed systems. *Twelfth IEEE International Symposium on High Performance Distributed Computing*, Seattle, U.S.A., 2003.
42. Wang M, Ramamohanarao K, Chen J. Trust-based robust scheduling and runtime adaptation of scientific workflow. *Concurrency and Computation*: *Practice and Experience* 2009; **21**:1982–1998.
43. Matossian V, Bhat V, Parashar M, Peszynska M, Sen MK, Stoffa PL, Wheeler MF. Autonomic computing: An overview. *Concurrency and Computation*: *Practice and Experience* 2005; **17**(1):1–26.
44. Rahman M, Hassan MR, Buyya R. Jaccard based availability prediction for enterprise grids. *Proceedings of the 10th International Conference on Computational Science* (*ICCS'10*), The Netherlands, May 2010.
45. Basney J, Livny M. Deploying a high throughput computing cluster. In *High Performance Cluster Computing*, vol. 1, ch. 5, Buyya R (ed.). Prentice-Hall: Elglewood Cliffs, NJ, 1999.
46. Abawajy JH. Fault-tolerant scheduling policy for grid computing systems. *Proceedings of 18th International Parallel and Distributed Processing Symposium* (*IPDPS'04*), U.S.A., April 2004.
47. Durschel P. The renaissance of decentralized systems. *Keynote Talk at the 15th IEEE International Symposium on High Performance Distributed Computing*, Paris, France, 2006.
48. Gupta KK, Nath B, Ramamohanarao K. Layered approach using conditional random fields for intrusion detection. *IEEE Transactions on Dependable and Secure Computing* 2010; **7**(1):35–49.
49. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems* 2009; **25**(6):599–616.
50. Ranjan R, Buyya R. Decentralized overlay for federation of enterprise clouds. *Handbook of Research on Scalable Computing Technologies*, Li K (ed.). IGI Global: Hershey, PA, U.S.A., 2009.
51. Auyoung A, Chun B, Snoeren A, Vahdat A. Resource allocation in federated distributed computing infrastructures. *Proceedings of 1st Workshop on Operating System and Architectural Support for the On-demand IT Infrastructure* (*OASIS'04*), U.S.A., October 2004.
52. Gong L. JXTA: A network programming environment. *IEEE Internet Computing* 2001; **05**(3):88–95.
53. Askalon TF. A tool set for cluster and grid computing. *Concurrency and Computation*: *Practice and Experience* 2005; **17**(2–4):143–169.
54. Foster I, Kesselman C, Tsudik G, Tuecke S. A security architecture for computational grids. *Proceedings of the 5th ACM Conference on Computer and Communications Security* (*CCS-98*), San Francisco, U.S.A., November 1998.
55. Rhea S, Godfrey B, Karp B, Kubiatowicz J, Ratnasamy S, Shenker S, Stoica I, Yu H. Opendht: A public dht service and its uses. *Proceedings of the ACM SIGCOMM 2005 Conference on Applications*, *Technologies*, *Architectures*, *and Protocols for Computer Communications*, August 2005.
56. Foster I, Kesselman C. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications* 1997; **11**(2):115–128.
57. Fitzgerald S, Foster I, Kesselman C, von Laszewski G, Smith W, Tuecke S. A directory service for configuring high-performance distributed computations. *Sixth IEEE International Symposium on High Performance Distributed Computing*, Portland, U.S.A., 1997.
58. Buyya R, Abramson D, Giddy J. Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid. *Proceedings of 4th International Conference on High Performance Computing in Asia–Pacific Region* (*HPC Asia 2000*), Beijing, China, 2000.

Copyright © 2011 John Wiley & Sons, Ltd.

59. Chapin S, Katramatos D, Karpovich J, Grimshaw A. The legion resource management system. *Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing* (*JSSPP'99*) *in Conjunction with the International Parallel and Distributed Processing Symposium* (*IPDPS'99*), San Juan, Puerto Rico, April 1999.
60. Tannenbaum T, Wright D, Miller K, Livny M. Condor—A distributed job scheduler. *Beowulf Cluster Computing with Linux*. The MIT Press: U.S.A., 2002.
61. Deelman E, Singh G, Su M, Blythe J, Gil A, Kesselman C, Mehta G, Vahi K, Berriman GB, Good J, Laity A, Jacob JC, Katz DS. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming* 2005; **13**(3):219–237.
62. Zhao Y, Wilde M, Foster I, Voeckler J, Dobson J, Gilbert E, Jordan T, Quigg E. Virtual data Grid middleware services for data-intensive science. *Concurrency and Computation*: *Practice and Experience* 2006; **18**(6):595–608.
63. Foster I, Vöckler J, Wilde M, Zhao Y. Chimera: A virtural data system for representing, querying, and automating data derivation. *Proceedings of the 14th International Conference on Scientific and Statistical Database Management* (*SSDBM*), Edinburgh, Scotland, July 2002.
64. Wu XF, Taylor V, Stevens R. Design and implementation of prophesy automatic instrumentation and data entry system. *Proceedings of the 13th IASTED International Conference on Parallel and Distributed Computing and Systems* (*PDCS'01*), U.S.A., August 2001.
65. Oinn T, Addis M, Ferris J, Marvin D, Senger M, Greenwood M, Carver T, Glover K, Pocock M, Wipat A, Li P. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics* 2004; **20**(17):3045–3054.
66. Stevens R, Robinson A, Goble C. mygrid: Personalised bioinformatics on the information grid. *Proceedings of 11th International Conference on Intelligent Systems for Molecular Biology*, Brisbane, Australia, July, 2003.
67. Taylor I, Shields M, Wang I. Resource management of triana p2p services. *Grid Resource Management*. Wiley-InterScience: The Netherlands, 2003.
68. ICT Businesses using Public Clouds. Available at: http://aws.amazon.com/solutions/case-studies/ [17 November 2010].
69. Armbrust M, Fox A, Griffith R, Joseph AD, Katz RH, Konwinski A, Lee G, Patterson DA, Rabkin A, Stoica I, Zaharia M. Above the clouds: A berkeley view of cloud computing. *Technical Report UCB/EECS-2009-28*, EECS Department, University of California, Berkeley, February 2009.
70. Varia J. Cloud architectures. *Technical Report*, Amazon Web Services, 2009.
71. Windows azure platform. Available at: http://www.microsoft.com/azure/ [17 November 2010].
72. Google app engine. Available at: http://code.google.com/appengine/ [17 November 2010].
73. Buyya R, Ranjan R, Calheiros RN. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. *CoRR*, abs/1003.3920, 2010.
74. Quiroz A, Kim H, Parashar M, Gnanasambandam N, Sharma N. Towards autonomic workload provisioning for enterprise grids and clouds. *Proceedings of the 10th IEEE/ACM International Conference on Grid Computing* (*Grid 2009*), Banf, AL, Canada, 2009; 50–57.
75. Ranjan R, Harwood A, Buyya R. Peer-to-peer-based resource discovery in global grids: A tutorial. *Communications Surveys Tutorials*, *IEEE* 2008; **10**(2):6–33.

Copyright © 2011 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe