A Taxonomy of Autonomic Application Management in Grids

Mustafizur Rahman[1], Rajiv Ranjan[2], and Rajkumar Buyya[1]

[1]Cloud Computing and Distributed Systems Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
{mmrahman, raj}@csse.unimelb.edu.au

[2]Service Oriented Computing Research Group
School of Computer Science and Engineering
The University of New South Wales, Australia
rajiv@unsw.edu.au

Abstract

In this paper, we propose a taxonomy that characterizes and classifies different components of autonomic application management in Grids. We also survey several representative Grid systems developed by various projects world-wide to demonstrate the comprehensiveness of the taxonomy. 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.

1 Introduction

Due to the establishment of Grid as a distributed and collaborative resource sharing environment, many of the large-scale scientific applications, such as workflows are currently executed in Grids. Thus, application management has emerged as one of the most important Grid services in past few years. An Application Management System (AMS) is generally employed to define, manage, and execute these scientific applications in Grid resources. However, the increasing scale complexity, heterogeneity, and dynamism of Grid environment that includes networks, resources, and applications have made such application management systems brittle, unmanageable, and insecure.

Autonomic Computing (AC) [21] is an emerging area of research for developing large-scale, self-managing, complex distributed 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.

This paper aims to survey the existing Grid systems that support autonomic application management. We classify these systems with respect to different aspects of autonomic application management, 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 implemented or incorporated in these systems.

The rest of the paper is arranged as follows. Section 2 presents the taxonomy that categorizes autonomic application management with respect to key features of AC. In Section 3, we map the proposed taxonomy onto selected Grid systems. We conclude the paper in Section 4.

2 Taxonomy

The taxonomy categorizes and classifies the approaches of autonomic application management in the context of computational Grids with respect to the key features of AC. 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 (see Fig. 1). In this section, we discuss each element and its classification in detail.

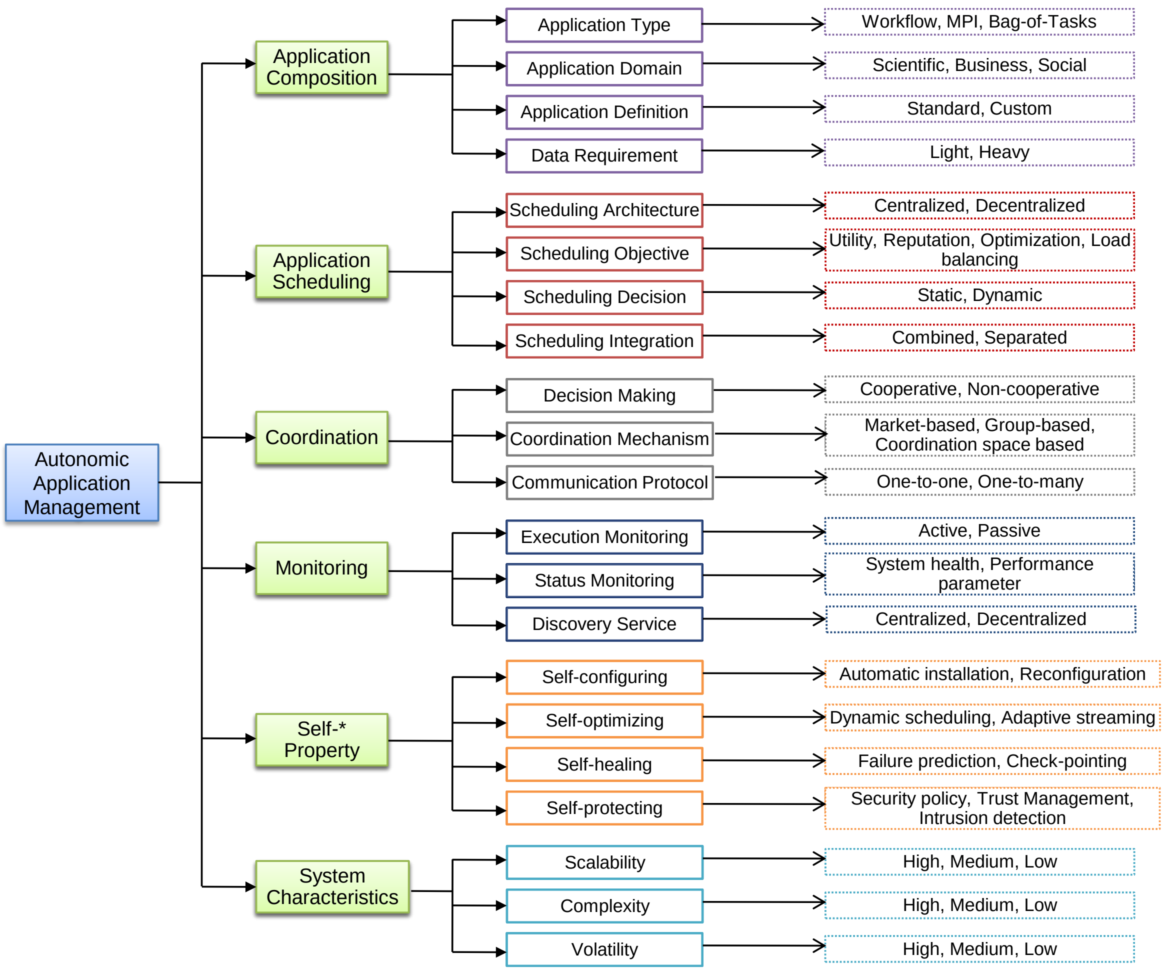

Figure 1: Taxonomy of autonomic application management for Grid computing.

2.1 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.

2.1.1 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.

A BOT application [10] 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 [20] are composed of multiple tasks, where inter-task communication is developed with Message Passing Interface (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 [25]. In a workflow, a task that does not have any parent task is called entry task, and a task that does not have any child task is called exit task. A child task can not be executed until all of its parent tasks are completed. The output of a workflow application is achieved when the exit tasks finish execution.

2.1.2 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) [18] 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) [1] 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.

2.1.3 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) [3] is widely used specially for workflow specification as it facilitates information description in a nested structure. Therefore, many XML-based application definition languages have been adopted in Grids. Some of these languages, such as WSDL [4], BPEL [17] have been standardized by the industry and research community (i.e. W3C [5]), whereas some of them are customized (i.e. xWFL [34], AGWL [12]) 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 for application definition, where the application composition is better visualized. However, this graphical representation is later converted into other forms for further manipulation.

2.1.4 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 feed 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.

2.2 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.

2.2.1 Scheduling Architecture

The architecture of scheduling infrastructure is very important with regards to scalability, autonomy, and performance of the system [16]. It can be divided into three categories: centralized, hierarchical, and decentralized.

In centralized scheduling architecture [34], 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.

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 [16]. 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.

2.2.2 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 endeavour 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 regards 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.

2.2.3 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 [32]. 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 [24] 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.

2.2.4 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 [33], 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.

2.3 Coordination

The effectiveness of autonomic application management in a distributed computing environment also depends on the level of coordination among the autonomic elements, such as application scheduler or resource broker, local resource management system, and resource information service. Lack of coordination among these components may result in communication overhead, which eventually degrades performance of the system. In general, the process of coordination with respect to application scheduling and resource management in Grids, involves dynamic information exchange between various entities in the system.

2.3.1 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 [14] 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, cooperative decision making approach [23] 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.

2.3.2 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 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 well-known agent coordination mechanism called contract net protocol [30] 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 Service Level Agreements (SLA).

In collaborative Grid environment, resource sharing is often coordinated by forming groups (e.g. Virtual Organization (VO) [13]) 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 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 [19] provides a global virtual shared space for the autonomic elements 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 [31][28] 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, while 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.

2.3.3 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.

2.4 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 is 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 autonomic element in Grids require appropriate monitoring data to recognize failure or suboptimal performance of any resource or service.

2.4.1 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 mange the unexpected events, such as failure. We identify two types of execution monitoring in Grids: active and passive.

The concept of active and passive monitoring of task execution in Grids is derived from the push-pull protocol [29], 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 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.

2.4.2 Status Monitoring

The autonomic elements in Grids needs to monitor the relevant system properties (i.e. system health) to optimize its operating condition and facilitate efficient decision making. System health data relates 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 and vmstat in Linux) or tools to perform this operation.

In addition, the autonomic element also needs to monitor the performance parameters of its operation, such as SLA violation if it incorporates market based mechanisms to interact with other autonomic elements in the system. To this end, it periodically measures the performance metrics (e.g. service uptime) with regards 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 [7] 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.

2.4.3 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 (GMD) [35]), monitoring data are stored in a centralized repository. Current studies have shown that [36] existing centralized model for resource directory services do not scale well as the number of users, brokers, and resource providers increase 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.

2.5 Self-* Properties

The evaluation of an autonomic system depends on to what extent it adopts or implements the self-* properties. Generally, its 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.

2.5.1 Self-configuring

Self-configuration refers to the ability to adapt to changes in the system. In regards to autonomic application management, the system can demonstrate 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.

2.5.2 Self-optimizing

Self-optimization refers to the ability to improve performance of the system through continuous optimization. In Grids, the application management systems can use dynamic scheduling techniques 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 utilization of adaptive streaming technique for data intensive applications [8].

2.5.3 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 [22] 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 [2] transfers the failed tasks transparently to other resources so that the task can continue its execution from the point of failure. The replication technique [6] executes the same task simultaneously on multiple Grid resources to increase the probability of successful task execution.

2.5.4 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 at both resource and user sides, such as providing accurate warning about potential malicious attack, taking networked resources offline if any anomaly is 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 [11]. 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 [15], 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.

2.6 System Characteristics

Due 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. The more a system becomes complex or volatile, the more it needs to incorporate autonomic computing principles in order to avoid performance degradation and user satisfaction. Whereas, increasing the scalability of a system facilitates itself to adopt autonomic features gracefully.

2.6.1 Complexity

According to the definition of Buyya et al. [9], 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' quality-of-service requirements. These 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.

2.6.2 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 peer-to-peer Grid systems [26] are scalable in nature, whereas the centralized Grid systems are least scalable as there exists a single point of control.

2.6.3 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 behaviour of the environment, such as configuration change, resource failure, and load variation. If the condition of the system changes frequently over 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.

Table 1: Summary of Grid projects

	Taxonomy breakdown	**Aneka Federation**	**Askalon**	**AutoMate**	**Condor-G**	**GWMS**	**Nimrod-G**	**Pegasus**	**Taverna**	**Triana**
Application composition	*Application type*	Bag-of-task	Workflow	Bag-of-task/ Workflow	Bag-of-task/ MPI	Workflow	Bag-of-task/ MPI	Workflow	Workflow	Workflow
	Application domain	Business/ Scientific	Scientific	Business/ Scientific	Scientific	Scientific	Business/ Scientific	Scientific	Scientific	Scientific
	Application definition	XML-based custom	AGWL-based custom	XML-based custom	ClassAd-based custom	xWFL-based custom	DPML-based custom	VDL-based custom	Scufl-based custom	Tool-based
	Data requirement	Light	Light/ Heavy	Heavy	Light	Heavy	Heavy	Heavy	Heavy	Light
Application scheduling	*Scheduling architecture*	Decentralized	Centralized	Decentralized	Centralized	Centralized	Centralized	Centralized	Centralized	Centralized/ Decentralized
	Scheduling objective	Load balancing	Utility/ Optimization	Load balancing	Load balancing	Utility/ Optimization	Utility/ Optimization	Optimization	Optimization	Optimization
	Scheduling decision	Dynamic	Static/ Dynamic	Dynamic	Dynamic	Dynamic	Dynamic	Static/ Dynamic	Dynamic	Dynamic
	Scheduler integration	Combined	Separated/ Combined	Combined	Combined	Separated	Separated/ Combined	Separated	Separated/ Combined	Separated/ Combined
Coordination	*Decision making*	Cooperative	Non-cooperative	Cooperative	Non-cooperative	Non-cooperative	Non-cooperative	Non-cooperative	Non-cooperative	Cooperative
	Coordination mechanism	Coordination space based	Market/ Group based	Coordination space based	Group based	Market/ Group based	Market/ Group based	Group based	Group based	Group based
	Communication protocol	One-to-many	One-to-one	One-to-many	One-to-one	One-to-one	One-to-one	One-to-one	One-to-one	One-to-many (all)
Monitoring	*Execution monitoring*	Passive	Passive	Passive	Active	Passive	Passive	Active	Passive	Passive
	Status monitoring	System health	System health	System health/ Performance parameter	System health	System health/ Performance parameter	System health/ Performance parameter	System health	System health	System health
	Directory service	Decentralized	Centralized	Decentralized	Centralized	Centralized	Centralized	Centralized	Centralized	Decentralized
Self-* properties	*Self-configuring*	Reconfiguration	N.A.	Reconfiguration	N.A.	N.A.	N.A.	N.A.	N.A.	Reconfiguration
	Self-optimizing	Dynamic rescheduling	Dynamic rescheduling/ Performance prediction	Adaptive streaming	Dynamic rescheduling	Dynamic rescheduling	Dynamic rescheduling	Dynamic rescheduling	Dynamic rescheduling	Dynamic rescheduling
	Self-healing	Failure detection	Failure detection/ Check-pointing	N.A.	Failure detection	Failure detection	Failure detection	Failure detection/ Check-pointing	Failure detection	Failure detection/ Check-pointing
	Self-protecting	N.A.	Authentication detection	Security policy	N.A.	N.A.	N.A.	Authentication detection	N.A.	N.A.
System characteristics	*Scalability*	High	Medium	High	Low	Low	Low	Low	Low	High
	Complexity	High	Medium	High	Low	Medium	Medium	Medium	Medium	High
	Volatility	High	Medium	High	Medium	Medium	Medium	Medium	Medium	High

3 Survey of Grid Systems

This section provides a detailed survey of selected existing Grid systems and mapping of the taxonomy proposed in previous section onto these systems. A comparison of various Grid systems and their categorization based on the taxonomy is shown in Table 1.

4 Conclusion

A taxonomy for autonomic application management in Grids has been presented in this paper. The taxonomy focuses on various aspects of autonomic application management, such as application composition, scheduling, monitoring, coordinating, and failure handling as well as system characteristics and self-management properties. In addition, we also survey some representative Grid systems and classify them into different categories using the taxonomy. Thus, this paper facilitates to understand the key features for autonomic application management and identify the possible future enhancements.

References

[1] Better business using grid solutions, bingrid. `http://www.beingrid.eu`.

[2] The directed acyclic graph manager, condor project. `http://www.cs.wisc.edu/condor/dagman/`.

[3] Extensible markup language (xml) 1.0 (third edition). `http://www.w3.org/TR/REC-xml/`.

[4] Web services description language (wsdl) version 1.2. `http://www.w3.org/TR/wsdl12`.

[5] World wide web consortium (w3c). `http://www.w3.org/`.

[6] J. H. Abawajy. Fault-tolerant scheduling policy for grid computing systems. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS04), USA*, April, 2004.

[7] S. Agarwala, Y. Chen, D. S. Milojicic, and K. Schwan. Qmon: Qos- and utility-aware monitoring in enterprise systems. In *Proceedings of the 3rd IEEE International Conference on Autonomic Computing (ICAC'06), Ireland*, June, 2006.

[8] V. Bhat, V. Matossian, M. Parashar, M. Peszynska, M. K. Sen, P. L. Stoffa, and M. F. Wheeler. Autonomic oil reservoir optimization on the grid. *Concurrency and Computation: Practice and Experience, vol. 17, no. 1, pp. 1-26*, 2005.

[9] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems, vol. 25, no. 6, pp. 599-616*, 2009.

[10] W. Cirne, F. Brasileiro, J. Sauve, N. Andrade, D. Paranhos, E. Santos-Neto, and R. Medeiros. Grid computing for bag of tasks applications. In *Proceedings of the 3rd IFIP Conference on E-Commerce, E-Business and E-Government, Brazil*, September, 2003.

[11] P. Durschel. The renaissance of decentralized systems. In *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC'06), France*, June, 2006.

[12] T. Fahringer, J. Qin, and S. Hainzer. Specification of grid workflow applications with agwl: An abstract grid workflow language. In *Proceedings of the 5th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'05), UK*, May, 2005.

[13] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications, vol. 15, no. 3, pp. 200222*, 2001.

[14] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-g: A computation management agent for multi-institutional grids. In *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC'01), USA*, June, 2001.

[15] K. K. Gupta, B. Nath, and K. Ramamohanarao. Layered approach using conditional random fields for intrusion detection. *IEEE Transactions on Dependable and Secure Computing, vol. 7, no. 1, pp. 35-49*, 2010.

[16] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of job-scheduling strategies for grid computing. In *Proceedings of the 1st IEEE/ACM International Workshop on Grid Computing (Grid'00), India*, December, 2000.

[17] M. B. Juric, B. Mathew, and P. Sarang. *Business Process Execution Language for Web Services.* Packt Publishing, UK, 2004.

[18] E. Laure and B. Jones. Enabling grids for e-science: The egee project. Technical Report EGEE-PUB-2009-001, CERN, Switzerland, 2009.

[19] Z. Li and M. Parashar. Comet: A scalable coordination space for decentralized distributed environments. In *Proceedings of the 2nd International Workshop on Hot Topics in Peer-to-Peer Systems (HOT-P2P'05), USA*, July, 2005.

[20] P. Nascimento, C. Sena, J. da Silva, D. Vianna, C. Boeres, and V. Rebello. Managing the execution of large scale mpi applications on computational grids. In *Proceedings of the 17th International Symposium on Computer Architecture on High Performance Computing (SBAC-PAD'05), Brazil*, October, 2005.

[21] M. Parashar and S. Hariri. *Autonomic Computing: An Overview, Unconventional Programming Paradigms, J.-P. Banatre et al. (eds.).* LNCS, Springer Verlag, Germanay, 2005.

[22] M. Rahman, M. R. Hassan, and R. Buyya. Jaccard based availability prediction for enterprise grids. In *Proceedings of the 10th International Conference on Computational Science (ICCS'10), The Netherlands*, May, 2010.

[23] M. Rahman, R. Ranjan, and R. Buyya. Cooperative and decentralized workflow scheduling in global grids. In *Future Generation Computer Systems, vol. 26, no. 5, pp. 753-768.* Elsevier Press, Amsterdam, The Netherlands, 2010.

[24] M. Rahman, S. Venugopal, and R. Buyya. A dynamic critical path algorithm for scheduling scientific workflow applications on global grids. In *Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing (eScience'07), India*, December, 2007.

[25] L. Ramakrishnan and D. Gannon. A survey of distributed workflow characteristics and resource requirements. Technical Report TR671, Indiana University, USA, September, 2008.

[26] R. Ranjan and R. Buyya. *Decentralized Overlay for Federation of Enterprise Clouds, Handbook of Research on Scalable Computing Technologies, K. Li et al. (eds.).* IGI Global, USA, 2009.

[27] R. Ranjan, M. Rahman, and R. Buyya. A decentralized and cooperative workflow scheduling algorithm. In *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'08), France*, May, 2008.

[28] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware'01), Germany*, November, 2001.

[29] A. Russo and R. L. Cigno. Push/pull protocols for streaming in p2p systems. In *Proceedings of the 28th IEEE International conference on Computer Communications (INFOCOM'09) Workshops, Brazil*, April, 2009.

[30] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers, vol. C-29, no. 12, pp. 1104-1113*, 1980.

[31] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, USA*, August, 2001.

[32] H. Topcuoglu, S. Hariri, and M. Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 3, pp. 260-274*, 2002.

[33] S. Venugopal, R. Buyya, and L. Winton. A grid service broker for scheduling e-science applications on global data grids. *Concurrency and Computation: Practice and Experience, vol. 18, no. 6, pp. 685-699*, 2006.

[34] J. Yu and R. Buyya. *Gridbus Workflow Enactment Engine, Grid Computing: Infrastructure, Service, and Applications, L. Wang et al. (eds.).* CRC Press, USA, 2009.

[35] J. Yu, S. Venugopal, and R. Buyya. A market-oriented grid directory service for publication and discovery of grid service providers and their services. *The Journal of Supercomputing, vol. 36, no. 1, pp. 17-31*, 2006.

[36] X. Zhang, J. L. Freschl, and J. M. Schopf. A performance study of monitoring and information services for distributed systems. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03), USA*, June, 2003.