API Specification for a Small Grid Middleware - SCADY

Rakesh Bhatnagar
MCA Department, S. K. Patel Institute of Management & Computer Studies
Kadi Sarva Vishwavidyalaya University
Gandhinagar, Gujarat 382023, India
rakesh_bhatnagar@rediffmail.com

Dr. Jayesh Patel
MCA Department, Acharya Motibhai Patel Institute of Computer Studies
Ganpat University
Kherva, Gujarat 382711, India
jayeshpatel_mca@yahoo.com

Abstract *- **In this fast and rapidly developing IT era, Grid computing is emerging and is being used in many industries and organizations. This has opened a window of attraction for research on Grid Architecture, Middleware and Services.***

Grid computing has resulted in faster execution of applications. Day by day the requirement for execution speed of applications is increasing. To achieve this suitable middleware is needed and to be configured requirement specific. An application can be executed faster only when there is no node failure or network failure. Moreover, at run time, if a node fails, there should be some alternative arrangement to reconfigure other node to complete the task which was supposed to be done by the failed node. Thus, a middleware is needed that can be configured according to the requirements of the application. Such a middleware would help in increasing the performance of the application.
In most of the middleware available presently, dynamic configuration is not facilitated. This has become one of the reasons for the failure of adoption or implementation of Grid in many organizations.

This paper proposes extended API specifications that can help in dynamic configuration of the middleware for execution of applications and efficient utilization of resources.

Keywords: Middleware, Dynamic Configuration, API Specification

I. INTRODUCTION

It is now a fact that the era of high performance computing has already started and is also taking leaps. Organizations are coming with new ideas and applications for their business which are complex. They also need these applications to execute faster and produce results.

Our focus area in this paper is Grid Computing. Grid computing is rapidly emerging as technology and infrastructure for wide area distributed computing. In wide area distributed computing, the resources are distributed globally and are used for computing. So the resources need to be properly shared and used. The security issues [1] should also be taken into consideration when sharing the resources.

In Grid computing, task is allocated and resources are managed through the resource manager. But as the number of resources is vast, the probability of failure of a resource is high. If the resources are not dynamically monitored and managed, the computation will fail. So the resources must be used efficiently and effectively. This is done by the middleware.

Another thing which is becoming significant is the use of peer to peer computing. Concept of centralized resources and database is gone. The computers share the resources directly saving time and effort.

But there are several factors that result in delaying of Grid adoption despite the enormous opportunity and value that the grid presents for companies and organizations.

In this paper, problem resulting due to static configuration of middleware are studied and an extension in API is proposed that can help in dynamic configuration of the middleware. Paper is organized as follows – Literature review is discussed in section 2. From the review, the problem is identified and a middleware is proposed in section 3. API specification for this middleware is given in section 4. In section 5, a case study is given that uses all the constructs of the API. Section 6 is the conclusion and section 7 contains the references.

II. LITERATURE REVIEW

Due to the dynamic nature of the Internet, node or network failures are the main challenges for grid computing. Moreover, as the nodes have autonomy, they may join or leave the grid system dynamically. We can conclude from this that a middleware is required that can ensure smooth working of applications in spite of node or network failure.

Although in inter related applications, dependability and reconfigurable criteria are very significant. Even if a single node fails due to some reason, the entire task and the computation can fail resulting in Grid failure. Failures and dynamic nature of load allocation and de-allocation implies that there is a strong requirement of applications or rather middleware that can dynamically adapt to the situation to improve the overall efficiency and throughput. However, to our best knowledge, existing grid technologies provide only limited reconfigurable criteria and scalability. We studied following systems –

- Unicore [7]
- Globus [8]
- Sun Grid Engine [9]
- Gnutella P2P Network [10]
- Freenet Network [11]

- Gridsim [12]

These systems provide static configuration of middleware or the application. Working of the Gnutella is given below as it is a fine example of peer to peer network file sharing. It throws the light on distributed computing. Almost the same is being followed in the other systems.

Working of Gnutella network as given in [10] -

- Users place the files they want to share on their hard disks and make them available to everyone else for downloading in peer-to-peer fashion.
- Users run a piece of Gnutella software to connect to the Gnutella network.
- There is no central database that knows all of the files available on the Gnutella network. Instead, all of the machines on the network tell each other about available files using a distributed query approach.
- There are many different client applications available to access the Gnutella network.

However, Gnutella has at least three disadvantages listed in [10]:

- There is no guarantee that the file you want is on any of the machines you can reach.
- Queries for files can take some time to get a complete response.
- Our machine is part of this network. It is answering requests and passing them along, and in the process routing back responses as well. Some amount of our bandwidth is given to handle requests from all the other users.

In [3], authors have identified that presently available middleware does not adequately address the configurability and re-configurability requirements of heterogeneous and changing environments. Current platforms cannot be configured to operate in diverse network types. And so they proposed a new middleware. But the issues of memory and flexibility are still to be worked upon.

In [4], authors have identified that new applications are being designed as per the requirements and have to be used with the existing deployed applications. This requires constantly changing processing power and communication requirement. And as grids span large geographic locations, number of failures and fluctuations of load become very critical. So authors have proposed a middleware infrastructure based on MPI implementation. This solution enhances the MPI implementations with reconfiguration capability and dynamic load balancing. Issues that are still pending are – mapping between application topologies and environment topologies, evaluation of load balancing procedures, support to non-iterative applications, solving complex applications, and merging the new applications with the existing ones.

In [14], authors have proposed a new component based lightweight middleware based on Service Component Architecture (SCA). The SCA eases the reconfiguration of the components at runtime to support different communication mechanisms and service discovery protocols. But this architecture is suitable only for mobile devices. And authors are still working upon the issues of memory and bandwidth for dynamic adaptation.

In [15], authors have identified that several middleware technologies have been designed and successfully used to support the development of stationary distributed systems built with fixed networks. Their success has been mainly due to their ability of making distribution transparent to both users and software engineers, so that systems appear as single integrated computing system.

However, completely hiding the implementation details from the application becomes more difficult and makes little sense in a mobile setting. Authors have concentrated on Mobile environment. Mobile systems need to quickly detect and adapt to drastic changes happening in the environment. A new form of awareness is needed to allow application designers to inspect the execution context and adapt the behavior of middleware accordingly.

Summing up, most of the existing grid middleware solutions provide re-configurability but to a limited extent. Many issues such as memory, bandwidth, processing load are still not solved completely. Hence there is a need of a powerful and flexible dynamic reconfigurable middleware that can serve the purpose.

III. PROPOSED SOLUTION

We are proposing a dynamically reconfigurable middleware SCADY (Scalable & Dynamic) which provides a dependable execution environment for grid applications. The middleware is divided into three layers which forms the core of SCADY. These layers are – initial layer, task layer, and a reconfigure layer.

We have used API concepts of Unicore [7], Globus [8] and GridSim [12]. We have tried to extend it in our middleware - SCADY. The studied environment is utilized for a case study. Some of the constructs that can help in providing dynamic configuration are given in next section. And a case study is given that uses these constructs.

The computation in our grid middleware is initiated by the user on the user node, who submits a grid task to one of the available grid nodes along with the task configuration details. The configuration details include, the number of nodes required, type of task and a metafile containing task input descriptions. The middleware determines the leader for the task based on the task id and then sends the task along with the configuration details to the leader, through the reconfigure layer.

First step for executing the subtask on remote grid node is to load the object. This can be done by using the getObject method. Remote grid node receives the object as a file. By using getObject method, it will convert it into the Object. Then it will execute the run method followed by callback methods on this object.

Whenever a node running a sub task, detects the failure of the other node, it notifies the leader through the reconfigure layer. The leader then stops all the individual sub-tasks by sending appropriate signals to the subtask nodes. The node which detected the failure in the first place (failed-node), then chooses a new node to restore the failed sub-task on. The new node is selected from the task layer. The failed node, then informs the leader of the selected node. The leader updates this information in its base and restores the computation by sending a signal to each of the sub task nodes.

IV. SCADY API SPECIFICATION

The API specification for the middleware and its components are discussed as under -

A. NODE MANAGER

This component acts as a mediator between user program and Scady Grid Middleware. It provides the basic constructs for configuring the grid middleware. The node manager is used by the application to specify the task definition, configuration details, and to get back the results. The main constructs of node manager are as follows.

- Void setScadyNodeIP(String IP): It is used by the application to set IP address of a live grid node. This node is mainly used for constructing the task overlay.
- Void setScadyMetric(Metric m): It is used to specify application metric, i.e. the way of measuring the capability of a node.
- Void setScadyQuery(Query q): It is used to specify the predicate or criteria for selecting the node.
- Void setScadyConditionType(ConditionType type): This construct indicates type of predicate used for selecting the node.
- Void Scadybarrier(): It is a blocking call. It will wait till all previously started remote sub tasks to finish. This method is analogous to the 'joinAll' method provided in Java.
- Void setScadySchedulerType(SchedulerType type): It is used to set the type of scheduler used for executing the application. By default it uses the dynamic scheduler.
- Void Scadyinitialize() ThrowsException: This call initiates with the given configuration requirements. It can throw three different types of exceptions. This method should be called before submitting any sub task.
- Void setScadyMinNodes(int min): It is used to set the minimum number of nodes required for execution. If middleware does not find those many capable nodes, it will not start the execution of an application.
- Void setScadyMaxNodes(int max): It is used to specify maximum parallelism, that is maximum number of parallel sub tasks.
- Void setScadyReplicaSize(int rno): It is used to set the middleware replication factor. It will indicate the maximum number of parallel node failures that the middleware can handle.
- Void setScadySurplusValue(int sno): It is used to set the number of surplus nodes. These surplus nodes are used to replace the failed nodes.
- Void Scadyclose(): It closes the node manager of Scady. In other words, it will remove all the temporary files, free all allocated buffers & close all open threads.
- Double getScadyHPF(): This construct is used for getting the HPF of a node.
- Void setScadyCPUFractionValue(double cpuFraction): This construct is used to set the CPU fraction used for calculating the HPF of a node.
- Void setScadyMemoryFractionValue (double memoryFraction): This construct is used to set the memory fraction used for calculating the HPF of a node.
- Void setScadyUpperBound(double value) & void etScadyLowerBound(double value): These two methods are used for setting the upper and lower bounds in query.

B. CONTEXT CLASS

This class is used for passing parameters to subtask. The subtask could be running on any grid node. The two important methods in this class are get and put. The syntax of these methods is as follows.

- Object Scadyput (String key, Object value): This construct is used for placing the 'value' into the table with given 'key'. It is used for passing inputs to the subtask.
- Object Scadyget (String key): This method used for getting the value. Before starting the execution of a subtask, it will retrieve the input parameters from the Context object.

C. RESULT CLASS

This class is used for transferring the results from the remote node to client program. Result class is similar to context. But difference is that context is used for passing the inputs while it is used for getting the results back. Another difference with respect to implementation is that the context does not support the user defined objects for storing and retrieving while result will support it. In addition, result will also support file transfers between

remote node and client node.

- Object Scadyput (String key, Object value): This construct is used for placing 'value' into the table with given 'key'. Value could be any predefined or user defined object. At the end of subtask completion, subtask puts its result into this Result object.
- Object Scadyget(String key): This method used for getting the value. Once the client receives subtask results, it will use get method for retrieving the results.
- Void ScadyputFile(ScadyFile): This construct is used for placing file into the result object.
- ScadyFile getFile(): This construct is used for getting file at the client node. If there is no file, it will return null.

V. CASE STUDY

A case study is done to write a program to find a cube of a number. The program is written in Java. The program is tested in 2 phases. First test was done in the institute and the department itself with 3 nodes, then with 10 and then 50 nodes. Nodes were having 2GB of RAM, 2.66GHz speed and Core 2 Duo processor.

Second test was done in the campus network of Kadi Sarva Vishwavidyalaya University. It has a wireless network that spans across 55 Kms covering four campuses.

A. PERFORMANCE CHARACTERISTICS -

The execution time was measured for all cases and is shown in Table-I and Table-II.

Table-I represents the facts gathered from the execution done inside the institution in MCA department i.e. local network with 3, 10 and 50 nodes. Table-II represents the facts gathered from the execution done outside the institution in the university network i.e. network with subnets with 3, 10 and 50 nodes.

TABLE-I
EXECUTION INSIDE THE INSTITUTE

Number of Nodes	Time (in Secs)	
	In other Middleware	In Scady Middleware
3	1.62	1.39
10	1.54	1.37
50	1.81	1.71

TABLE-II
EXECUTION IN THE CAMPUS NETWORK

Number of Nodes	Time (in Secs)	
	In other Middleware	In Scady Middleware
3	3.25	3.07
10	3.48	3.28
50	4.77	4.52

These figures are gathered in the peak working time. Alchemi middleware was used (As other middleware) with SCADY middleware. From the figures, we can rightly identify that as compared to Alchemi, performance of Scady middleware in terms of execution speed is better.

VI. CONCLUSION

In this paper, we studied middleware and their APIs and found that even though, most of them succeeded in providing sufficient performance in grid computations, it is not easy to provide a dynamically configurable middleware that can help in ensuring that the computation does not fail even if any node fails. To solve this problem, we have constructed a small grid middleware – SCADY and tried to add and extend the existing API of the available middleware. We can conclude that if the APIs are extended and functionality is given, then middleware can be dynamically configured resulting in high throughput and faster execution speed in the computation.

VII. FUTURE SCOPE

Scady middleware should be tested in different environments. It should be tested even when the load in the network is considerably less. Performance measurement of Scady should also be done with other middleware such as Unicore, Globus, etc. A scenario also should be created in which deliberately a node should be made to fail and then the middleware should be tested for dynamic reconfiguration.

REFERENCES

[1] Rakesh Bhatnagar, Dr. Jayesh Patel, "An Empirical study of Security issues in Grid Middleware", In- ternational Journal of Emerging Technology & Advanced Engineer- ing , ISSN 2250–2459, Volume 4 Issue 1, Jan 2014, Pages 470-474

[2] M. Venkateswara Reddy, A. Vijay Srinivas, Tarun Gopinath, and D. Janakiram, "Vishwa: A reconfigurable P2P middleware for Grid Computations", http://dos.iitm.ac.in/LabPapers/icppVishwa.pdf

[3] Hirunavukkarasu Sivaharan, Gordon Blair, and Geoff Coulson, "GREEN: A Configurable and Re-configurable Publish-Subscribe Middleware for Pervasive Computing ", On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE, Lecture Notes in Computer Science Volume 3760, Springer 2005, pp 732-749

[4] Kaoutar El Maghraouia, Travis Desella, Boleslaw K. Szymanskia, James D. Terescob, and Carlos A. Varela, "Towards a Middleware Framework for Dynamically Reconfigurable Scientific Computing", Elsevier, Advances in Parallel Computing, Volume 14, 2005, Pages 275–301 and http://www.cs.rpi.edu/~szymansk/papers/hpc.04.pdf

[5] http://www.academia.edu/6142596/A_component based_middleware_framework_for_configurable_and_reconfigurable_Grid_computing

[6] Middleware definition, retrieved on Jan 10, 2014, from: *www.cl.cam.ac.uk/teaching/1011/C DSysII/12-middleware.pdf*

[7] Unicore API Documentation- http://www.unicore.eu/documenta- tion/api/unicore6/

[8] Globus API Specification – http://toolkit.globus.org/toolkit/doc s/5.0/5.0.0/execution/gram5/pi/

[9] Sun Grid Engine API Specification– http://***gridengine****.eu/program-ming-****apis***

[10] How Gnutella Works – HowStuff- Works, http://computer.howstuffworks.co m/file-sharing2.htm

[11] The Freenet Project - *https://****freen****- ****et****project.org/*

[12] Gridsim API Specification - http://www.cloudbus.org/gridsim/doc/api/gridsim/package-

summary.html

[13] Nagaraja R., Murali Bhaskaran V., “Dynamically reconfigurable middleware for pervasive computing DOMPC”, http://hdl.handle.net/10603/23948, PhD Thesis.

[14] R. Nagaraja, Dr. G. T. Raju, “COMMPC - Component Based Middleware for Pervasive Computing”, IJCSNS International Journal of Computer Science and Network Security, VOL.11 No.9, September 2011

[15] L. Capra, W. Emmerich, C. Mascolo, "Middleware for Mobile Computing: Awareness vs. Transparency", HOTOS, 2001, Proceedings of 8th Workshop on Hot Topic in Operating Systems, 2001, pp. 0164

[16] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf "Achieving Expressiveness and Scalability in an Internet-Scale Event Notification Service", 19th ACM Symposium on Principles of Distributed Computing (PODC2000), Portland OR. July, 2000

[17] Coulson, G., Blair, G.S., Clark, M., Parlavantzas, N., “The Design of a Highly Configurable and Reconfigurable Middleware Platform”, ACM Distributed Computing Journal, Vol 15, No 2, pp 109-126, April 2002.

[18] Blair, G., Coulson, G., Grace, P., "Research Directions in Reflective Middleware: the Lancaster Experience", Proceedings of the 3rd Workshop on Reflective and Adaptive Middleware (RM2004) co-located with Middleware 2004, Toronto, Ontario, Canada, October 2004

[19] Clark, M., Blair, G.S., Coulson, G., Parlavantzas, N., “An Efficient Component Model for the Construction of Adaptive Middleware”, Proc. IFIP Middleware 2001, Heidelberg, Germany, Nov. 2001.

[20] Kon, F., Costa, F., Blair, G.S., Campbell, R., "The Case for Reflective Middleware: Building Middleware that is Flexible, Reconfigurable, and yet simple to Use", CACM, Vol. 45, No. 6, pp 33-38, 2002.

[21] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan, “Chord: A scalable peer-to-peer lookup service for internet applications”, SIGCOMM '01 Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, Pages 149-160.

[22] Gianpaolo Cugola, Gian Pietro Picco, “REDS: a reconfigurable dispatching system”, SEM '06 Proceedings of the 6th international workshop on Software engineering and middleware, Pages 9 – 16

[23] Kaoutar El Maghraoui, Travis J. Desell, Boleslaw K. Szymanski, and Carlos A. Varela, “The Internet Operating System: Middleware for Adaptive Distributed Computing”, International Journal of High Performance Computing Applications, Vol. 20, No. 4, Winter 2006, pp. 467-480

[24] Carlos A. Varela, Paolo Ciancarini, Kenjiro Taura, “Worldwide computing: Adaptive middleware and programming technology for dynamic Grid environments”, Journal Scientific Programming - Dynamic Grids and Worldwide Computing, Volume 13 Issue 4, October 2005, Pages 255-263

[25] Kaoutar El Maghraouia, Boleslaw K. Szymanski, and Carlos A. Varela, "An Architecture for Reconfigurable Iterative MPI Applications in Dynamic Environments”, Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science Volume 3911, 2006, pp 258-271, http://link.springer.com/chapter/10.1007/11752578_32

[26] Yuhui Deng, Frank Wang, Na Helian, Sining Wu, Chenhan Liao, “Dynamic and scalable storage management architecture for Grid Oriented Storage devices”, Elsevier, Parallel Computing, Volume 34, Issue 1, January 2008, Pages 17–31

[27] T. Purusothaman, S. Annadurai, H. Vijay Ganesh, C.T. Chockalingam and B. Uthra Kumar, “Dynamically Scalable, Heterogeneous and Generic Architecture for a Grid of Workstations”, Journal of Grid Computing (2004) 2: 239–24, Springer 2005

[28] Geoffrey Fox, Shrideep Pallickara, and Xi Rao. A scaleable event infrastructure for peer to peer grids. In JGI '02: Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande, pages 66.75, New York, NY, USA, 2002. ACM Press.