Introducing High Availability in Threads based Grid Middleware Architecture

Khurram Ali Shah
Software Engineering
UET Taxila, Pakistan

Ali Javed
Assistant Professor
UET Taxila,Pakistan

Muhammad Irfan
Software Engineering
UET Peshawar

ABSTRACT
The architecture proposed in "A Novel Grid Middleware" is a light weight, easy to implement and an open grid computing architecture in terms of allowing organizational/personal computing resources to participate in the grid when their computing resources are free. The architecture was based on allocation, distribution and execution of the threads in the grid. The current architecture does not have a solution on the high availability of the grid. This research is an extension to the earlier proposed architecture [1] and will make the architecture to answer the real world problems of high availability and will make it an implementable solution. This research is providing solution to the scenarios that how can threads based grid can ensure high availability in case threads are lost during their execution or the resources of the grid becomes unavailable for any reasons. This research provided the architecture design on introduce high availability in the threads based grid so that in case at any point of time a thread has stopped responding, how the redundant thread can be made available to ensure continues execution of the grid based application.

General Terms
Threads, Grid, Middleware, Distributed Computing, Quality of Service, Connectivity, Main program/thread, Redundant program/thread.

Keywords
TDAC Server (Thread Distribution And Allocation Server), TDAC Client (Thread Distribution And Allocation Client), HASynchronizer (High Availability Synchronizer), Checkpoints and Replica.

1. INTRODUCTION
The proposed grid middleware architecture in previous paper [1] considered ideal conditions (no thread/connectivity/resource loss) and described the basic workflow of our grid middleware. In this research work, we want to extend our previous architecture in terms of Quality of Service. We will introduce such techniques that will ensure high availability of grid resources and how grid middleware will respond if a thread is lost or connectivity of resources is down or malfunctioning in processing a foreign thread.

In brief we will develop data/process flow charts that will prove the research work and we also will develop architecture for answering how the architecture can be utilize by the programmers.

We are Introducing two additional features. The first one is Checkpoints in the Program/ Thread. The second feature is simply Introducing a replica of the same program/thread . High Availability Synchronizer is responsible to synchronize the data and instruction execution between Main Program/Thread and Replica Program. These Building blocks (HA Synchronizer, Program/Thread, Replica Program/Thread) are connected together in such a way that Application requests HA Synchronizer for threads execution. HA Synchronizer is communicating with both actual Program/Thread and Replica Program/Thread in order to sync the data when checkpoint is reached. HA Synchronizer is placed in TDAC server to implement the enhanced architecture.

2. RELATED WORKS
Evolution of Grids started from the fields such as the Internet, distributed computing, web services, a number of cryptography technologies given that security measures and virtualization technology. These technologies were present and used for various purposes. The grid technology combines these technologies and takes advantages of features of these fields to develop such a computational environment that provides computational resources to perform various tasks. These tasks could be stock exchange simulations and studies for accurate oil and gas exploration, scientific research like forecast of weather or serving corporate/business requirements for an organization whose existence is geographically dispersed. The showbiz industry makes severe use of computers to provide movies using a growing number of animations.

The objective of Grid Middleware is to provide users invisible computing ability and fair access to the computational resources [2] in a grid environment which is heterogeneous. The architecture of gird middleware should be such that it could be easily reused, robust [3] so that it requires minimal time to develop and deploy it again. It is a combination of software application and API's that generates users and allow them to use the resources of gird system.

When computational, storage, network and scientific instruments combines together to form grid fabric of grid [4]. The resources could be computers clusters, servers and even supercomputers existing at many physically distributed organizations. Scientific instruments could be thermostats of nuclear plans, sensor networks telescopes that provide real-time data which might be transmitted directly to computational sites or that could be stored in any database. Now to use those resources in an effective manner, we need middleware's which provides the resources for computation while protecting the critical data and resources of the resource providers.

The middleware provides abstraction of network complexities from the developers. Robust and easy useable middleware design is very useful. [5]. It should reduce the development and deployment time of it again. Globus Toolkit (GT) is a

very significant middleware and it is a de facto touchstone for grid implementation. There are two similar incorporate middleware products [6] that are UNICORE and Legion. They offer the most significant components looked-for in grid middleware. UNICORE [7] is the resultant of the project UNICORE (UNiform Interface to COmputing REsources), which has shaped the technology that presents faultless, protected, and instinctive access to distributed grid resources. Converting UNICORE into a gird service is done in Unicore/GridService [8].

Legion is a theoretical foundation for a metasystem [9]. The Legion object permits the carrying out of jobs in heterogeneous platforms. Applications written in various programming language can be executed as Legion provides interoperability among objects developed in multiple languages [10].

Condor is an application system that generates a HTC environment. It efficiently uses of the processing cycles of idle workstations in a network. Condor makes available a job queuing method, scheduling procedure, priority plan, resource observing, and resource management [11].

3. INTRODUCING HIGH AVAILABILITY OF THREADS

In this section, the proposed model for ensuring high availability of threads will be discussed and different scenarios for high availability [12] of the grid will be explored. We will also elaborate the data flow diagrams of the architecture to show the implementable solution.

3.1 Building blocks of introducing High Availability

Every Program/thread has 2 parts (Data, Instructions). In order to ensure high availability and prevent thread loss we are Introducing two additional features. The first one is Checkpoints in the Program/ Thread. It is a number or title pointing out that the certain instruction [13] has reached during the execution of the thread and now it is the time to sync its data with the thread initiator. It can be based on the logical sequencing of the program.

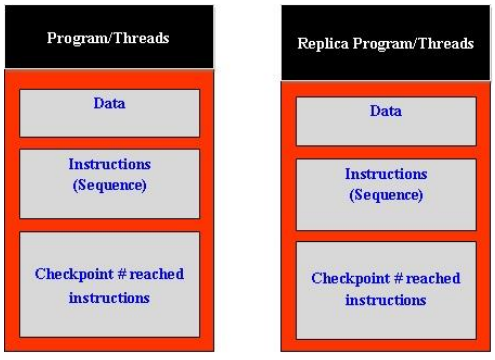

Figure 1. Replica Program/Thread

The second feature is simply Introducing a replica of the same program/thread . Checkpoints are also present in Replica program which will sync the data with the thread initiator. In case main program/thread is loss due to any reason, the Replica program/thread will continue from the last reached checkpoint.

3.2 Introducing High Availability Synchronizer (HASynchronizer)

High Availability Synchronizer is responsible to synchronize the data and instruction execution between Main Program/Thread and Replica Program and checking the availability of the main program and redundant program/thread. It has 5 sub modules which are shown in diagram and discussed below.

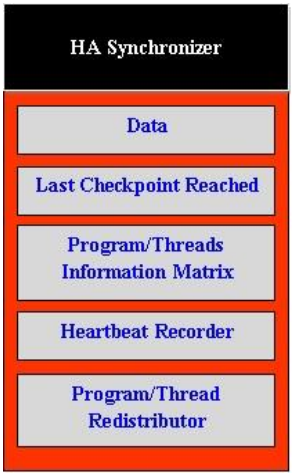

Figure 2. High Availability Synchronizer

Data: Keep updating data portion of the Program/thread with itself.

Last Checkpoint Reached: Recording the checkpoint reached in the program/thread of main and replica.

Program/Threads Information Matrix: Keep the information about the systems on which main and replica program are executed along with the program/thread information[14].

Heartbeat Recorder: Keep checking the status of the main or replica program/thread , its system/network availability where main or replica program/thread is executed.

Program/Thread Redistributors:

Re-distribution module is activated in case a thread loss happen or for any reason the systems executing the main or replica program are not reachable.

3.3 Connecting Building Blocks

Above discussed building blocks (HA Synchronizer, Program/Thread, Replica Program/Thread) are connected together in such a way that Application requests HA Synchronizer for threads execution [15]. HA Synchronizer is communicating with both actual Program/Thread and Replica Program/Thread in order to sync the data when checkpoint is reached.

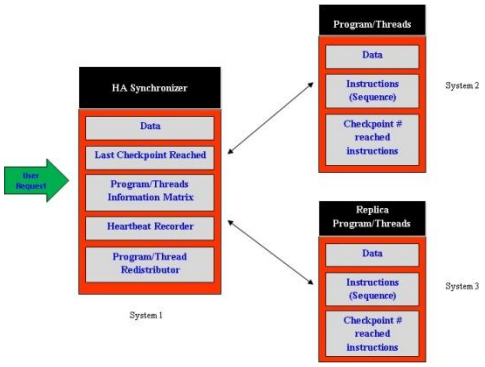

Figure 3. Connecting the Building Blocks

3.4 High Level Data Message Flow

Data message flow is shown in following diagram with detailed elaboration below.

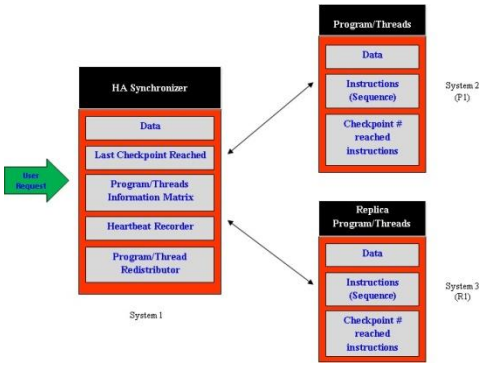

Figure 4. Data Flow Diagram

i. P1 shares the checkpoint info and updated data.
ii. R1 shares the checkpoint info and updated data.
iii. HA Synchronizer will keep updating the data and checkpoint info from P1, R1 and will re-distribute the program/thread to other systems [16] in case of main program or replica program is not available.

3.5 Enhanced TDAC Server Architecture

HA Synchronizer is placed in TDAC server to implement the enhanced architecture. Checkpoints Creator is added in Application with TDAC Client so that checkpoints are created in the thread.

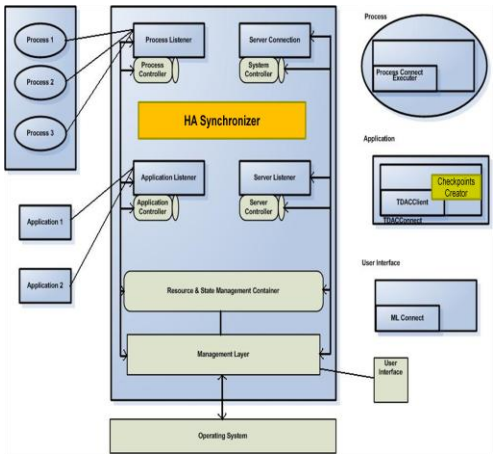

Figure 5. Enhanced High Availability Architecture

3.6 Incorporating the HA building blocks in the architecture

Server to server communication in presence of HA Synchronizer takes place between two TDAC Servers in the following diagram.

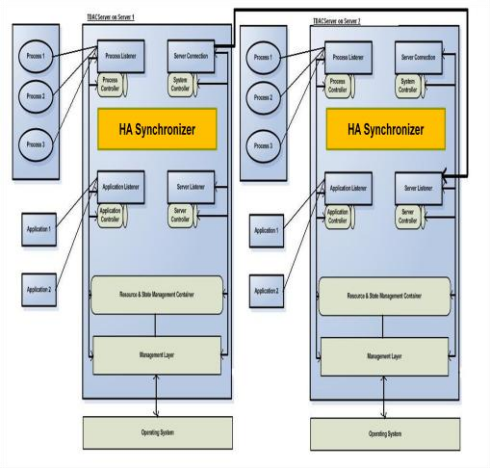

Figure 6. Incorporating the HA Synch in Enhanced Architecture

3.7 Simulating Enhanced Architecture

Now a complete flow of the enhanced architecture is shown in which a replica thread is executed simultaneously and HA Synchronizer is also present. Note that now communication between thread [17] and process occurs through TDAC server because HA Synchronizer is present in TDAC Server.

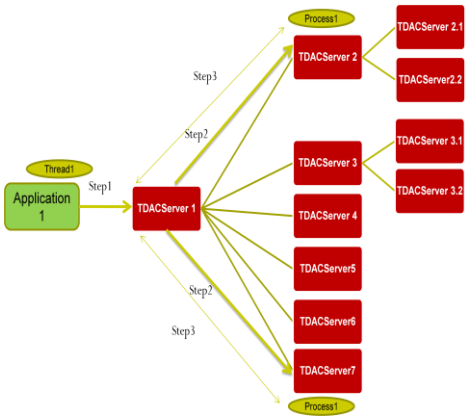

Figure 7. Simulating Enhanced Architecture

Following steps describe the detail of high availability grid:

Step1: Application1 Requesting TDACServer1 To Allocate Worker Process

Step2: TDACServer1 Requesting TDACServer2 and 7 To Allocate Worker Process For Execution of Application1's Thread1 (main and Replica).

Step3: TDAC1 will send Thread1 (Main and Replica) to Process1 of TDACServer2 and 7 respectively for execution.

Step4: TDAC1 will keep synchronizing with TDAC2 And TDAC7 as per earlier flows explained to Meet high availability requirement.

4. DATA FLOW AND INTER-ACTION BETWEEN THE BUIL-DING BLOCKS

In this section we will describe the data flow and Interaction between the different building blocks, discussed in previous section, of High Availability Architecture.

4.1 Initialization

The data flow diagram below shows the Starting of Main Program/Thread and Redundant Program/Thread. HA Synchronizer send an initialize message to Main and Redundant Program/Thread, which in return send and acknowledgement. Then HA Synchronizer send another message to share checkpoint details and update the result which is again acknowledged by the Main and Redundant Program/Thread.

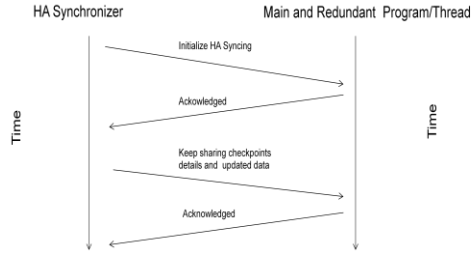

Figure 8. Initialization between HA Sync and Main/Redundant Thread

4.2 Checkpoint Arrived

Here Data Sharing is shown when New Checkpoint is Arrived and how data is being sync with the HA Synchronizer. When a checkpoint is reached, the Main and Redundant Program/Thread send a message to HA Synchronizer about the details of checkpoint. HA Synchronizer in return send a message to share data. As a result, the Main and Redundant Program/Thread send the latest data and HA Synchronizer send back an acknowledgement of data received.

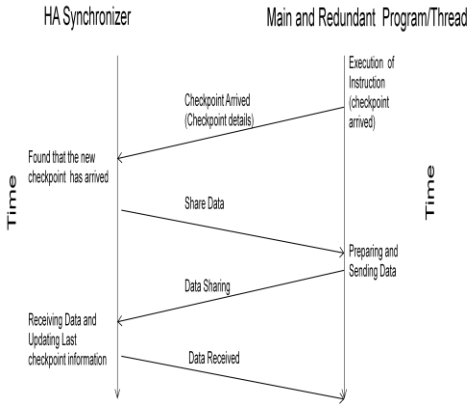

Figure 9. Checkpoint arrived

4.3 Checkpoint Already Updated

Now Data Sharing is shown when Checkpoint is Already Updated by Either Main or Redundant program/Thread. When Main or Redundant program/thread send checkpoint details to the HA Synchronizer and sync found that this checkpoint is already updated, then it sends a message to the program/thread that this data already exists. After that, main/redundant program continue execution of instructions.

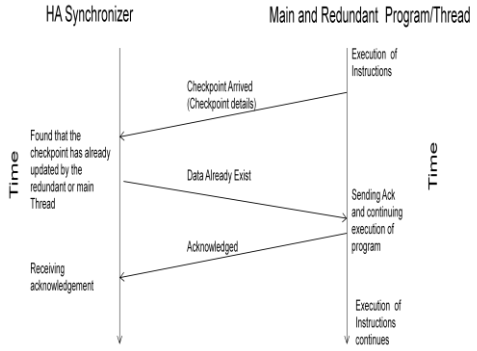

Figure 10. Checkpoint already updated

4.4 Heartbeat Recorder

HA Synchronizer will periodically send message to Main and Redundant Program/Thread or Systems for checking their availability. In return, Main and Redundant Program/Thread will send an acknowledgement so that HA Synchronizer will update Heartbeat Recorder Log. If acknowledgement is not received in certain time then the Redistributor module is activated which is discussed in next part.

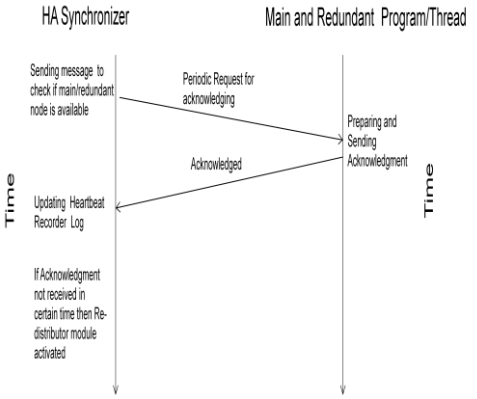

Figure 11. Heartbeat Recorder

4.5 Redistribution In Case Of Thread Loss

In last part, Message Passing to redistribute the program/thread to new systems is shown in case Main or Redundant Thread is lost. HA Synchronizer selects available resource after finding that main/redundant thread has lost and request the new system to recreate thread. New system sends acknowledgement and then HA Synchronizer encapsulate thread, last checkpoint information and data, and send package to the new system. Hence the new system starts the execution of thread onward from the last checkpoint.

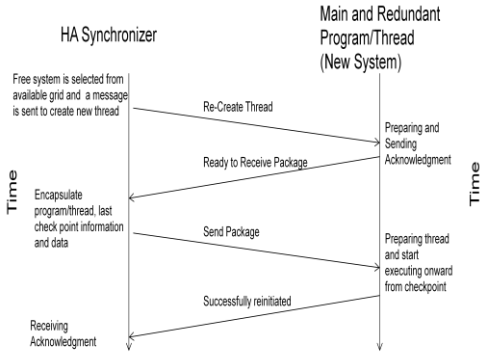

Figure 12. Redistribution in case of Thread Loss

5. CONCLUSION AND FUTURE WORK

Our architecture provides a solution for thread distribution, allocation and execution of application's thread in the grid. It is very easy to configure and any system/resource can be added at any point in our architecture. We have provided an administrative layer over Operating system.

We have introduced High Availability Synchronizer and Replica Program/Thread to ensure high availability of threads in our Grid. Now there is proper mechanism to handle high availability issues such as thread loss, any resource goes down and connectivity loss. In this way the proposed architecture is more implementable and handles the real world problems.

Security and User Interface are the open research fields in this proposed architecture and various methodologies can be used to ensure the security and User Interface for this architecture. Practical implementation of this architecture can open new gates for research in Grid middleware. We recommend java for its implementation because it is platform independent and provides easy way for remote threads creation and distribution.

6. REFERENCES

[1] Muhammad Asad Khan, Khurram Ali Shah, Muhammad Irfan. "A novel grid middleware architecture". FIT '10 Proceedings of the 8th International Conference on Frontiers of Information Technology. Published in ACM DL 2010..

[2] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: enabling scalable virtual organizations, volume 2150 of Lecture Notes in Computer Science, pages 200–222. Springer, 2001. http://www.globus.org/alliance/publications/papers/anatomy.pdf

[3] Gregor von Laszewski and Kaizar Amin. Grid Middleware, chapter Middleware for Communications, pages 109–130. Wiley, 2004. Available online at: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--grid-middleware.pdf

[4] Andrew S. Grimshaw, Wm. A. Wulf, and the Legion team. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39–45, 1997.

[5] Gregor von Laszewski and Kaizar Amin. *Grid Middleware*, chapter Middleware for Communications, pages 109–130. Wiley, 2004. Available online at: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--grid-middleware.pdf

[6] Parvin Asadzadeh, Rajkumar Buyya, Chun Ling Kei, Deepa Nayar, and Srikumar Venugopal. Global grids and software toolkits: a study of four grid middleware technologies. Web Published, 2004. Available online at: http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0407001

[7] UNICORE. UNICORE objectives. Web Published, 2007. Available online at: http://www.unicore.eu/unicore/

[8] David Snelling. UNICORE and the open grid services architecture, pages 701–712. Wiley, 2003.

[9] Parvin Asadzadeh, Rajkumar Buyya, Chun Ling Kei, Deepa Nayar, and Srikumar Venugopal. Global grids and software toolkits: a study of four grid middleware technologies. Web Published, 2004. Available online at: http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0407001

[10] Legion. The Legion project. Web Published, 2007. Available online at: http://legion.virginia.edu/

[11] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the Condor experience. *Concurrency and Computation: Practice and Experience*, 17(2–4):323–356, 2004.

[12] A Goldchleger, F Kon, A Goldman. InteGrade: object-oriented Grid middleware leveraging the idle computing power of desktop machines. http://www.researchgate.net/publication/220105482_InteGrade_object-oriented_Grid_middleware_leveraging_the_idle_computing_power_of_desktop_machines/file/d912f50c63abdc873e.pdf

[13] Hassan, K ; Abbes, H. ; Jemni, M. "From desktop grid to cloud computing based on BonjourGrid middleware". Electrical Engineering and Software Applications (ICEESA), 2013 International Conference.

[14] L Young, S McGough, S Newhouse. Scheduling architecture and algorithms within the ICENI Grid middleware, http://www.lesc.ic.ac.uk/iceni/downloads/materials/AHM2003/scheduling.pdf

[15] Hasanzadeh, M. Meybodi, M.R. "Deployment of gLite middleware: An E-Science grid infrastructure" Electrical Engineering (ICEE), 2013 21st Iranian Conference.

[16] E Caron, F Desprez, D Loureiro. Cloud computing resource management through a grid middleware: A case study with DIET and eucalyptus http://hal.archives-ouvertes.fr/docs/00/43/57/85/PDF/RR-7096.pdf

[17] S Gorlatch, J Dünnweber. From grid middleware to grid applications: Bridging the gap with HOCs http://wiki.ci.uchicago.edu/pub/VDS/DslCS/HOCAbstract/HOCbasics.pdf

Khurram Ali Shah has done MS in Software Engineering from University of Engineering and Technology Taxila Pakistan. He is working as Software Quality Assurance engineer at Altair Technologies Islamabad. His research interest includes Grid Computing, Data Communication, Software Project Management, Software Quality Assurance and Software Testing.

Ali Javed is serving as an Assistant Professor in the Department of Software Engineering at UET Taxila, Pakistan. He is also a PhD Scholar in Computer Engineering Department at UET Taxila, Pakistan. He has received his MS degree in Computer Engineering from UET Taxila, Pakistan in February, 2010. He received Chancellor's Gold Medal in MS Computer Engineering degree and became the first MS student in the history of UET Taxila to be awarded

Chancellor's Gold Medal. He has received B.Sc. degree in Software Engineering from UET Taxila, Pakistan, in September, 2007. He got 3rd position in Software Batch-2003F in BS Software Engineering degree. His areas of interest are Digital Image Processing, Computer vision, Video Summarization, Grid Computing, Mobile Application Development, Software Requirements Engineering, Software Quality Assurance and Software testing.

Muhammad Irfan is currently serving as Senior Service Quality and Standards Analyst in Al Jazeera Media Network. He is graduated from University of Engineering and Technology in Computer Software Engineering. His area of interests are Grid Computing, Project Management and Service Management.