Volume 3, No. 4, April 2012
Journal of Global Research in Computer Science
ISSN-2229-371X
RESEARCH PAPER
Available Online at www.jgrcs.info

FAULT TOLERANCE IN GRID COMPUTING USING WADE

G.Veeranjaneyulu[1], Srimathi.C[2]
School of Electronics Engineering, VIT University, Vellore, Tamilnadu-632014
Veeranji11@gmail.com[1], srimathic@vit.ac.in[2]

Abstract: Grid computing is a coordinated resource sharing and solving the problems in organizations which are dynamic and virtual in nature. Apart from the dynamic nature of grids which means that resources may enter and leave the grid at any time, in many cases outside of the applications, control grid resources are also heterogeneous in nature. Many grid applications will be running in environments where interaction faults are more commonly occur between diverse grid nodes. As resources may also be used outside of organizational boundaries, it becomes iteratively difficult to guarantee that a resource being used is not malicious one. Because of the diverse faults and failure conditions developing, deploying, and executing long running applications over the grid remains a challenge. Hence fault tolerance is an primary factor for grid computing. The prototype system is designed using agents to provide service replication, reactivation and avoids the single point of failure. The agents and the workflows are provided by a common software platform called WADE.

INTRODUCTION

Computational grids become increasingly important to solve computationally intensive and time consuming problems in science, industry, and engineering. Since the failure probability increases with a rising number of components, fault tolerance is an essential characteristic of parallel systems. Such systems must provide redundancy and mechanisms to detect and localise errors as well as to reconfigure the system and to recover from error states[1].

This project proposes to provide an information management framework as a flexible and reliable distributed service system which are applied for specific data processing tasks. This design is based on a station server unit which host dynamic services. The station servers are dynamically interconnected and provide a distributed framework for hosting services.

Unfortunately the task of building Grid applications remains extremely difficult because there are few tools available to support developers. To build reliable and re-usable Grid applications, programmers should be equipped with a set of tools that hides the details of most Grid services and provides developers with a consistent, non-complex model in which applications can be composed from well tested, reliable subunits. For some time, Grid workflows have been emerging as an important alternative to develop Grid applications. Grid workflow management systems are evolving towards a system that will offer both the definition of applications at a high level of abstraction (without dealing with implementation details) and their automatic deployment and execution on the basis of the available resources in the Grid. Moreover, the use of agents have been proved a good means for intelligent tasks distribution and for supporting users in both on-line and offline activities[2,4].

The system is designed using agents to provide service replication, reactivation and avoids the single point of failure. The agents and the workflows are provided by a common software platform called WADE (Workflows and Agents Development Environment).This helps in managing the complexity of the distribution both in terms of administration and fault tolerance. This fault tolerance can be increased using agent replication providing redundant copies of the agent, increasing rate of fault tolerance [2,5].

THE WADE PLATFORM

WADE (Workflow and Agent Development Environment) represents the main evolution of Jade, a popular open source middleware conceived to facilitate the development of distributed applications based on the agent-oriented paradigm. JADE provides a distributed runtime environment, the "agent" and "behaviour" abstractions, peer to peer communication between agents and basic agent lifecycle management and discovery mechanisms. WADE adds to JADE the support for the execution of tasks defined according to the workflow metaphor and a number of mechanisms that help managing the complexity of the distribution both in terms of administration and fault tolerance[3].

WADE comes with a development environment called "WOLF" that facilitates the creation of WADE-based application. WOLF is an Eclipse plug-in and as a consequence allows WADE developers to exploit both Eclipse IDE and WADE-specific features[3]. WOLF, the development environment for WADE-base applications. It has been implemented as a plug-in integrated in the Eclipse Platform. This choice allowed developing a complete environment to manage the whole life cycle of workflows. The integration with the Eclipse Platform allows also the exploitation of all features offered by the Eclipse Java IDE. WOLF represents a key element in the challenge to bring the workflow approach from the business process level to the level of system logics[3,8].

a. The Boot Daemon process: a process running in each host of the platform that it is in charge of the Containers activation within its local host.
b. The Configuration Agent (CFA): an Agent runnin in the Main Container and is responsible for interacting with the boot daemons and controlling the application life cycle
c. The Controller Agents (CA): there is a controller agent for each container in the platform and they are responsible for supervising activities in the

local container and for all the fault tolerance mechanisms provided by WADE.

d. The Workflow Engine Agents (WEA): They are the most distinguished components of a wade based application. In fact, instead of providing a single powerful workflow engine, WADE gives to each JADE agent the possibility of executing workflows and the Workflow Engine Agents are the agents enabled to the workflows executions, by means of an embedded micro workflow engine [5].

As mentioned above, WADE makes available both the expressiveness of a visual representation of workflows and the power of usual programming languages. It provides a graphical representation of a workflow with a well defined structure. Wade adopts the meta model defined by the XPDL[4], standard specified by the Workflow Management Consortium. This meta-model XPDL language conceived as interchange formalism between different systems. WADE supports the import of XPDL files and the adoption of this meta-model facilitates these operations. In particular, WADE supports all the elements required by the version 1.0 of the XPDL specification and some elements, related to the events, from the version 2.0.

In the XPDL metamodel a process is represented as a workflow, consisting of one or more activities that can be thought as tasks to be executed. In a workflow, the execution entry point is defined, specifying the first activity to be performed; this activity is called Start Activity. A workflow must have one or more termination points, named Final Activities.

The execution flow is defined by means of transitions. A transition is an oriented connection between two activities and may have a condition associated. Excluding the final ones, each activity may have one or more outgoing transitions. When the execution of an activity is terminated, the conditions associated to its outgoing transitions are evaluated. As soon as a condition is verified the corresponding transition is activated and the execution flow proceeds towards the destination activity.

Normally, a process execution uses some internal data5, for instance, to pass intermediate results between activities and/or for evaluation of conditional expressions. In the XPDL meta-model internal data are modelled by the Data Fields. A process can have one or more inputs to be provided and one or more outputs expected at the end of its execution. Inputs and outputs of a process can be formalized in the XPDL meta-model by means of the workflow Formal Parameters.

Fault Tolerance:

Fault tolerance [4,7] is a very important issue especially when dealing with real-world application. Using a distributed approach (as typically happens when building an application on top of JADE/WADE) provides a mean to deal with unexpected HW faults. JADE already includes suitable fault tolerance mechanisms that allow the platform to survive to a fault of a container or host. Such mechanisms however work at the platform level and not at the application level. That is they ensure that the mechanisms provided by the platform, such as message delivery and agent creation/destruction, continue to work even after an unexpected fault of one of the hosts where a JADE-based application is running. Of course if an agent implementing a given application specific piece of functionality F was running on the crashed host, the application (even if its remaining agents will still be able to exchange messages and exploit other platform services) may not work anymore as functionality F is no longer available.

In order to support fault tolerance at the application level, WADE provides an additional mechanism that enable automatically restarting all agents that suddenly disappear due to HW or SW faults. This mechanism is implemented by the Control Agents., each container (but the Main Container) holds a Control Agent that is responsible for supervising the resources in the local container. Furthermore Control Agents coordinate themselves so that a single leader is elected.

The “autorestart[4]” mechanism works as follows.

a. If an agent suddenly dies (this may happen if there is a software bug in the agent’s code that causes an uncaught exception), the Control Agent of the local container automatically restarts it.
b. If an entire container suddenly dies (e.g. because the container process is killed), of course the local Control Agent dies too. The leader Control Agent then takes care of restarting the whole container (this operation is actually performed by the BootDaemon on the host where the container was active upon a request from the leader Control Agent) with all its agents.
c. If the dead container included the leader Control Agent, other Control Agents elect a new leader that takes care of restarting the dead container.
d. If the recreation of the whole container fails (this is always the case when a container disappeared due to a fault of the underlying HW) the leader Control Agent restarts all agents that where living in the dead container through the Runtime Allocator Agent. The latter will then recreate all dead agents according to its agent allocation policies.

It should be noticed that the auto restart mechanism provided by WADE is only responsible for restarting the dead agents passing them the same arguments they were launched with. Application developers are responsible for implementing application specific mechanisms to restore the internal state of the agents after a fault/restart. In order to facilitate that, the WadeAgentImpl class provides the getRestarted() method that can be used in the agentSpecificSetup() method to distinguish between a normal startup and a restart after a crash.

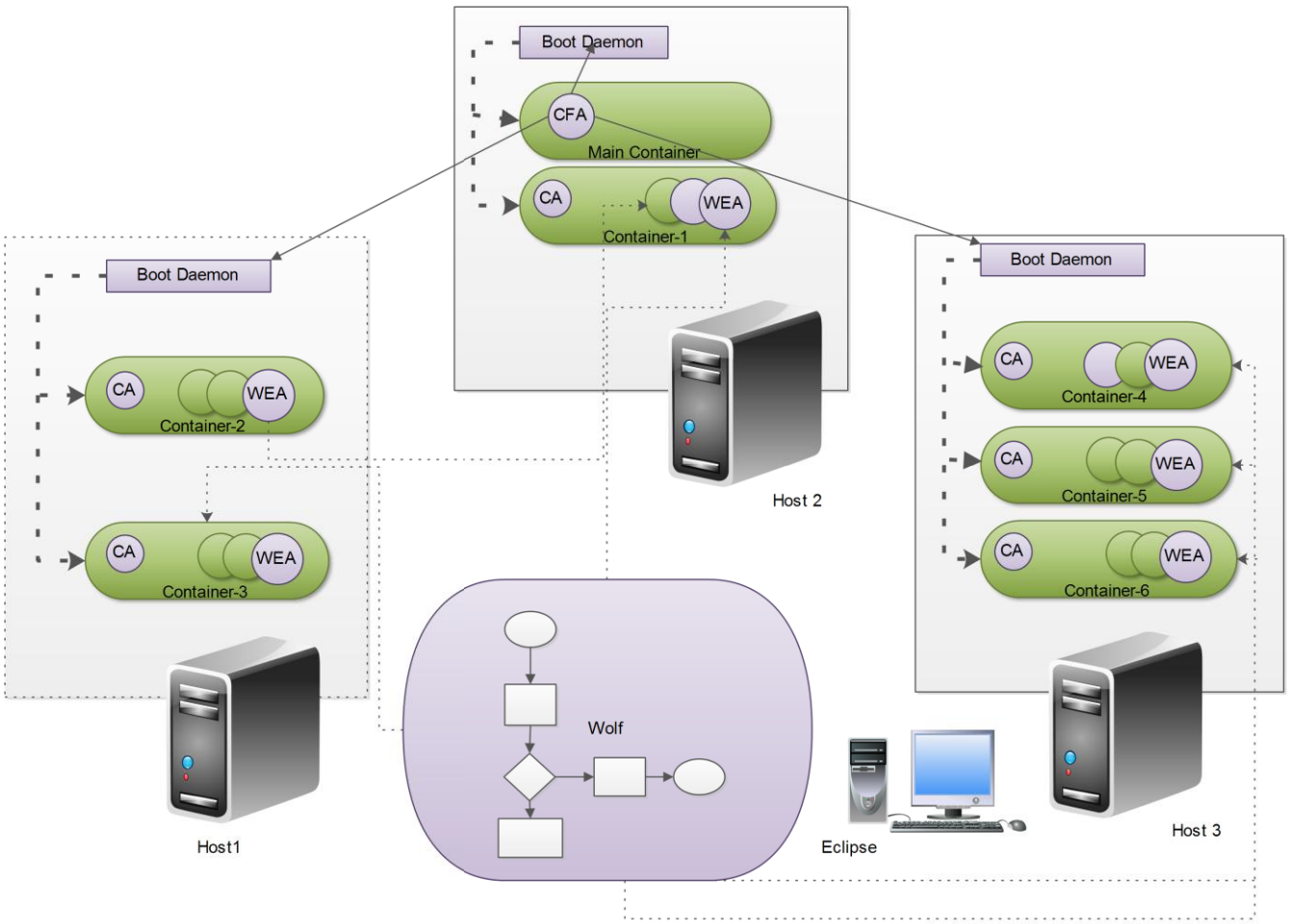

Figure 1. WADE architecture

SYSTEM DESIGN AND IMPLEMENTATION

Station server is a network service that can host dynamic services. These are interconnected providing a distributed framework for services. Each station server registers in a set of lookup servers by themselves and they acts as a remote listener in other station servers for any state modifications. Each station server gets a proxy for all other station servers and updates dynamic list of active station servers. The mechanism which is provided by the WADE platform automatically inform the other servers if any changes occur in other services and if any problems occur in the network[2].

The dynamic services which are hosted by the station servers made available to the interested clients. It allows the client services to access the information they require from the entire system. The remote event notification supports the communication between the dynamic services hosted by station servers and management. The remote event subscription of the services to register for transition events at the time of registration.

Agents are the dynamic services which can move between station servers to perform a certain task requested by the clients. Agents can interact in both synchronously or asynchronously using station servers. The station server does the service management and facilitates the inter-service communication.

A Wade based application consists of Main Container hosting the JADE, AMS and DF must be activated first with other containers registering to it at bootstrap time. WADE specific components are the boot daemon processes (one per host) responsible for activating containers in the local host. The Configuration Agent always running in the main container to interact with the boot daemons and controlling the application life cycle. Controller Agents (one per container)responsible for supervising activities in the local container and fault tolerant mechanisms. The workflow Engine Agents are able to execute the tasks in order to provide domain specific features.

The services which are provided by the server must be registered in the lookup server. The interested clients wants to access any service from the server, the request must check up in the lookup server whether the service is providing by the server or not, if the service is present in the lookup server then the request is transferred to the station server and the service can be accessed by the client through remote event notification and agents between them[4].

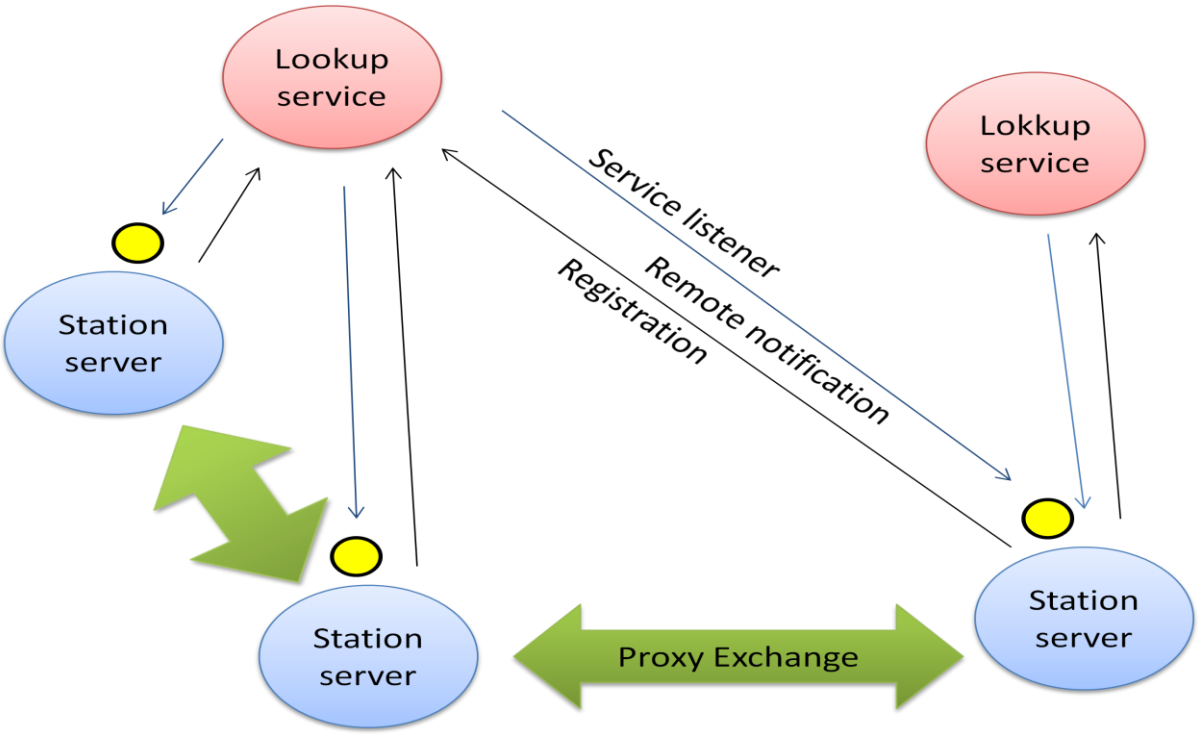

Figure 2. System Design

The services can be registered in the WADE by using DF registration. Agent identity is contained within an Agent Identifier (AID), composed of a set of slots that comply with The structure and semantics defined by FIPA. The name of an agent is a globally unique identifier that JADE constructs by concatenating a local name to the platform name. The agent addresses are transport addresses inherited by the platform, where each platform address corresponds to an MTP end point where FIPA-compliant messages can be sent and received. Agent programmers are also allowed to add their own transport addresses to the AID. AMS is the contact point for all agents that need to interact in order to access the white pages of the platform as well as to manage their life cycle. DF is the agent that implements the yellow pages service, used by any agent wishing to register its services or search for other available services. It also accepts subscriptions from agents that wish to be notified whenever a service registration or modification is made that match some specified criteria.

METHODOLOGY

Considering 4,8,16,32 server system designs, holding 20 services. In four server system design each server can hold five services in them. In eight server system design each server can hold four services. In sixteen server system design each server can hold three services. In thirty two server system design each server can hold two services. The Services are registered in the lookup server using Agent Identifier (AID), when a client requested for a service it verifies whether the service is available or not in the main server. If the service is available the request sent to the main server and the response for the request sent to the client through the control agents. Here system is designed for 4, 8, 16, 32 servers separately, to know how many replications are made in the local servers. The main server is hosting in main Container and all other local servers are working on different containers. Due to some server crashing or any failure occur in the main server, the request should not struck there itself, the request should send to the local servers managed by Configuration Agent, where the request is send to all servers in which replications for the services. After finding the requested service, response sent to the client. From this replication times are calculated and plotted the graphs for time taken searching the replications from client request and time taken from replications to reach client response.

RESULTS AND ANALYSIS

The replications of services for 32 server system design are more where more number of servers crashed also, the system can sustain the failure and the response for a request can sent to the client. For the 16 server system design the replication of services are reduced it is not that much efficient compared to 32 server system. For 8 server and 4 server system design replication of services are majorly reduced, here the probability of system failure is high if any failures occur in the system.

The Fig3shows the replications for 4, 8, 16, 32 server system designs. The blue colour indicates the first replication, the red colour indicates second replication and green colour indicates third replication in all server system designs. When a failure occurs in the system, time taken to find the replications increases when the number of servers are increasing in the system. The access to service is very fast while main server is failed, in the four server system design compared to other system designs, But the system is not reliable and efficient compared to other system designs because only single replication is available and is not tolerant to faults and there is no proper load handling. To

overcome this problem, eight server systems is designed, here two levels of replication of services is considered. This improves fault tolerance and load balancing between replications compared to four system designs. To handle more number of requests a sixteen server system design and thirty two server system can be used where fault tolerance is very high in both the systems and also load balancing is good compared to eight server system design. The disadvantage of thirty two server system design is taking more time to find replications and to give response to the client.

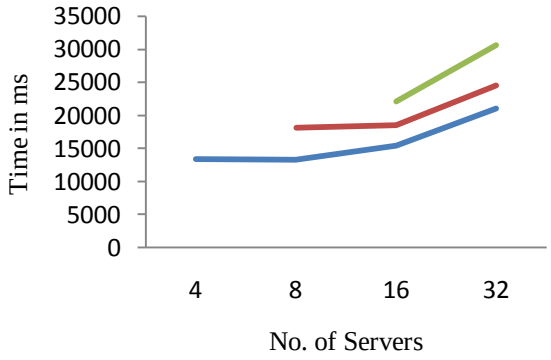

Figure 3. No of servers' vs. time taken to reach replications from client

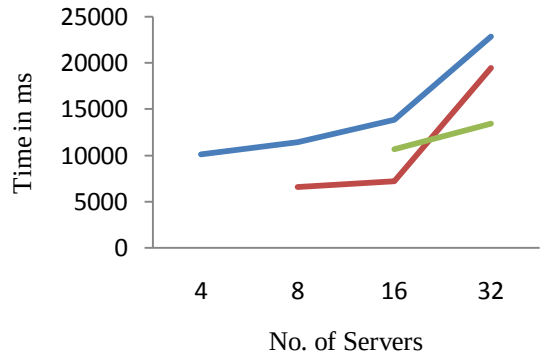

Figure 4. No of servers' vs. time taken to reach client from replications

From fig4 says that as number of replications increases time taken to respond to a client decreases i.e.; for an eight server system time taken to respond are less when compared to a four server system. It can also be inferred from the fig4 that as number of server's increases from sixteen to thirty two is more as number of replications increases and most of the time is wasted on finding the replicate.

CONCLUSION

This implementation is based on a multi-agent society built on WADE a software framework to aid the creation of agent applications in compliance with the FIPA specifications for interoperable intelligent multi-agent systems. Agent technology enables our platform to ensure high flexibility in defining and modifying operational processes, high control and maintainability of the logics used in and high performance, fault tolerant and scalability. From the analysis which is made on a WADE tool to implement a fault tolerant system in grid computing, it can be said that among all the system designs available a sixteen server system design is more preferable for handling the requests, load balancing and fault tolerance.

REFERENCES

[1] S.Siva Sathya,K.SyamBabu "Survey of FaultTolerant Techniques For Grid".

[2] Harvey B. Newman, Iosif C. Legrand , and Julian J. Bunn , "A Distributed Agent Based Architecture for Dynamic Services," California Institute of Technology, Pasadena, CA 91125, USA .

[3] FabioBelly Famine, Giovanni Caire, Dominic Greenwood, "Developing Multi-Agent Systems With Jade".

[4] Alessandro Negri, Agostino Poggi and Michele Tomaiuolo," Intelligent Task Composition and Allocation Through Agents," Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE'05) ,2005

[5] Kaizar Amin, Gregor von Laszewski, Mihael Hategan, Nestor J. Zaluzec, Shawn Hampton, and Albert Rossi, "GridAnt: a Client-Controllable Grid Workfow System", in *Proc.Hawaii International Conference on System Science (HICSS-37),* Island of Hawaii, HI, 2004.

[6] Elankovan Sundararajan, Aaron Harwood and Ramamohanarao Kotagiri "Incorporating Fault Tolerance with Replication on Very Large Scale Grids", in Eighth IEEE International Conference on Parallel and Distributed Computing, Applications and Technologies, 2007, pp. 319-328.

[7] Perakath C. Benjamin Charles Marshall Richard J. Mayer," A WORKFLOW ANALYSIS AND DESIGN ENVIRONMENT (WADE)" Proceedings of the 1995 Winter Simulation Conference,pp. 597-603

[8] JADE :http:jade.//tilab.com/wade/.