Research Journal of Applied Sciences 9 (3): 154-159, 2014
ISSN: 1815-932X

Fault Tolerant Middleware Framework to Improve QoS in Distributed Systems

[1]A. Samydurai and [2]A. Shanmugam
[1]Department of CSE, Dhanalakshmi College of Engineering, Chennai, Tamil Nadu, India
[2]Bannari Amman Institute of Technology, Sathyamangalam, Tamil Nadu, India

Abstract: In distributed computing environment there are several servers and quite a few number of middleware. This concept takes into account the load, object's memory usage, object's function execution duration and CPU usage of the middleware in handling remote method calls thus providing fault tolerance and improving QoS. Traditionally, the middleware performs an analysis over the parameters to ascertain the resource availability and to entrust the task to the appropriate server. So far, due consideration has not been given to the identification of the right middleware to manage such remote method invocations. This study focuses in particular on choosing an appropriate middleware for a given scenario and also performing sufficient data replication at chosen middleware to improve high availability and fault tolerance of data thus improving QoS of the distributed systems. Choosing the middleware with deliberate attention to vital parameters of QoS can impact in significant performance improvements and high reliability can be achieved through fault tolerance than merely being tied to the one prefixed by default.

Key words: Distributed systems, fault-tolerance middleware, Quality of Service (QoS), reliability, appropriate

INTRODUCTION

Distributed computing is a potentially powerful approach for accessing large amounts of computational power. Distributed computing refers to computational decentralization across a number of processors which may be physically located in different components, subsystems, systems or facilities (Armendariz *et al*., 2006). These processors may be general-purpose computers with data/application sharing capabilities, they may have an architecture that enables collaborative processing focused on a specific task and/or each may be optimized to efficiently execute particular tasks or control specific subsystems. The processors may include small microprocessor, workstations, minicomputers and large general purpose computer systems. A distributed system must provide various mechanisms for process synchronization and communication for dealing with the deadlock problem and for dealing with a variety of failures that are not encountered in a centralized system (Goel and Buyya, 2006). Building a distributed system is really more difficult and expensive than building a centralized one. The programming in heterogeneous distributed environment is made difficult by the following factors:

- Exchange of complex data
- Different encoding of data types
- Synchronization and real parallelism
- Need for atomic sequence operations

So, a middleware is needed in a distributed system environment. Middleware is a set of common business-unaware services that enable applications and end users to interact with each other across a network. In essence, middleware is the software that resides above the network and below the Business-Aware Application Software (Poza *et al*., 2009). Object Oriented Middleware (OOM) offers synchronous, typed communication between components of a distributed program. Developed out of a need to extend the object-oriented programming paradigm to distributed systems, the middleware typically consists of a mechanism to allow methods to be invoked on remote objects, plus services to support the naming and location of objects in a system-wide manner. Examples of OOM are Java RMI/JINI and CORBA. Both scientific and business applications today are generating large amount of data. Typical applications such as high energy physics and bioinformatics will produce petabytes of data per year. In many cases data may be produced or required to be accessed/shared at geographically distributed sites. Sharing of data in a distributed environment gives rise to many design issues, e.g., access permissions, consistency issues, security (Murray *et al*., 1998). The reason for such a widespread interest is due to following facts: increased availability, increased performance and Enhanced reliability (Natarajan *et al*., 2000).

As distributed systems become more and more a house hold technology and also support mission critical

Corresponding Author: A. Samydurai, Dhanalakshmi College of Engineering, Anna University, Chennai, India

application systems, the reliability of these systems becomes a crucial issue. When researchers talk about reliability of middleware in distributed systems, two contradicting facts become apparent. Since, middleware systems are distributed in more than one system, if there is single point of failure, the services can still be availed from other middleware or component. But since there is more number of components, the probability of failure of any single components also increases. When the system developers provide transparent system services in a distributed system, unless special care has been taken, there is a higher chance of failure of serving clients relying on a particular middleware (Dumitras and Narasimhan, 2007).

Traditionally, CPU usage, bandwidth, queue length of waiting instructions of the server and priority of the task determines the server to be deputed for the call received. The middleware performs an analysis over the above parameters to ascertain the resource availability and to entrust the task to the appropriate server. Remote method calls are scrutinized through a prefixed middleware and then it gets directed to the appropriate server hosting the remote object that encapsulates the method. This process is done with the intent of ensuring optimal resource utilization and speedy completion of tasks. So, far due consideration is not been given to the identification of the right middleware to manage such remote method invocations.

Researchers have proposed a new framework to improve QoS in distributed systems. The proposed framework combines fault-tolerance of middleware and replication of data on the server side for high availability and better performance to improve the reliability and QoS of distributed system. The proposed framework analyses various parameters in each of the available middlewares and find the one which can finish the present task within the shortest time. Fault tolerance is intrinsic in the proposed solution because, the middleware that consumes above normal computation time are not assigned any task. The parameters probed are the load, object's memory usage, object's function execution duration and CPU usage of the middleware in handling remote method calls.

LITERATURE REVIEW

Geol and Buyya (2006)'s replication is used in distributed environment. Here, data is stored at more than one site for performance and reliability reasons. Replication algorithms for distributed storage are discussed. A replication strategy suitable for a certain application or architecture is not suitable for other. The important difference in replication protocols is consistency requirements. If strict consistency is followed, performance is reduced. If it uses read only queries than performance can be increased. Natarajan *et al.* (2000) describe the patterns into the DOORS fault-tolerant CORBA used to improve its performance and fault-tolerance. The fault tolerant CORBA defines a standard set of interfaces, policies to provide high reliability. The FT-mechanism used to detect and recover from failure. Fault detectors used for detecting faults. Armendariz *et al.* (2006) discuss about the development in the replication concept of architecture innovations. Researchers require different replication strategies and business services. For this researchers must choose a efficient protocol for the middleware. It also presents new middleware architecture for enhancing the availability, fault tolerance and consistency of business information systems in the framework of a distributed infrastructure as part of the solicited project CONFIA.

Marculescu *et al.* (2003) have proposed pre-copying with remote execution as a novel. Their technique can be part of a design method for extending the lifetime of variety of applications under various types of faults despite scarce energy resources. Improving code migration and remote execution is done by Pre-Coping with Remote Execution (PCRE) that derives ideas from both. Kim (2001) conversed some major issues in realization of fault tolerance in RT OO DC Systems and give a new decade, the Real-Time (RT) computing application market is no longer a negligible market even for major platform vendors. This scheme explains us the potential and issues of objects oriented real time programming. There are many faulty that develop in middleware-kervel-hardware platforms. Technically middleware approach is tougher but enjoys a related position with others. They were discussed various techniques in RTOO System that helped in constituting a cost effective technology for the faulty tolerance realization.

Schantz and Schmidt (2002) construct new computing and information systems. Middleware was invented in an attempt to help simplify the software development of large-scale, network-centric computing systems and bring those capabilities within the reach of many more developers than the few experts at the time who could master the complexities of these environments. Distributed Object Computing (DOC) has emerged as a set of software protocol and service layers that help to solve the problems. Murray *et al.* (1998) describes a middleware technology which aims to realize this promised benefit of distributed systems by offering high availability abstractions. High availability is an application-dependent issue. Some rsault automates process replication, failure

detection and failure recovery. It can be plugged into an ORB to realise the goal of replication transparency, at least on the client side. It has two major iterations of algorithm design, implementation and optimisation. Gokhale *et al*. (2004) provides three contributions to the study CORBA middleware for performance-sensitive DRE Systems. They evaluate strategies for FT-CORBA help for mission-critical DRE Systems a reality. CORBA middleware provide an overview of FT-CORBA. DRE System focuses on developing and deploying standard CORBA middleware whose performance guarantees to applications even even faults occurs.

Kim *et al*. (2006) proposes is going to work upon the semantic middleware architecture to reason out the problem of errors. Its cause and solutions to its solving in unicom environment. Researchers have created models in ontology to describe service faulty and their functionalities. Three overarched are defined that breaks the service for better flexibility and scalability. The use of faulty tolerant semantic middleware called wapee gives a meaning to the service and fault ontology in faulty application. Poza *et al*. (2009) proposed the structure in distribution system which talks about the communication or interaction of QoS levels with different control layers. Researchers talked about the component application of the middleware in the control system. They also mentioned the drawbacks and the latest application components in the system.

QOS OF DISTRIBUTED SYSTEMS

This study elaborates the framework that focuses on improving the QoS of distributed systems. The proposed framework combines the fault-tolerance of object-oriented middleware and the replication of data on the server side. The proposed framework analyses various parameters in each of the available middleware and find the one which can finish the present task within the shortest time. This process ensures that the task initiated always gets directed to the middleware with comparatively minimum amount of load and has the best of computing power. Analyzing the computation time that will normally be consumed for the waiting objects in a queue not just resolves the middleware that can fulfill the request with the minimum waiting time but also inherently resolves the differences in computing power of individual middleware.

This is because the time that is required for computing depends primarily on the computing power of the analyzed middleware and hence it is ensured that the middleware with the best computing power and least amount of waiting instruction in queue will win the chance to receive the task initiated. Figure 1 depicts the general architecture of distributed systems with multiple middleware's.

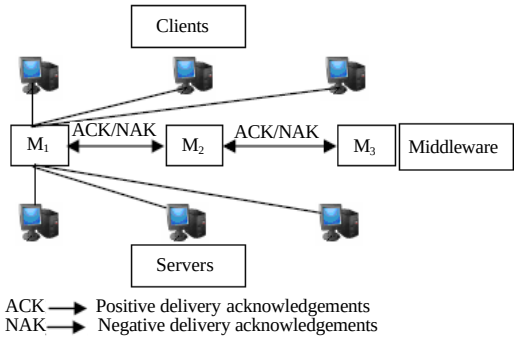

Fig. 1: General architecture of distributed systems with multiple middlewares

QoS metrics: A set of metrics are framed for the middleware operation.

Fault tolerance: The process of the middleware to withstand the parameter faults of the middleware.

Response time: The time taken by the middleware to service the request of the client and the server.

Monitoring: The ability of the middleware to monitor the transmission and reception of data between servers in the system.

IMPROVING FAULT-TOLERENCE OF MIDDLEWARE

Here, researchers concentrate on raising the fault tolerance aspect of middleware with an objective of improving QoS of the system. The algorithm proposed here identifies the apt middleware by scrutinizing vital parameters and improves the fault tolerance. The parameters probed are the load, object's memory usage, object's function execution duration and CPU usage of the middleware in handling remote method calls.

Client logs: Client logs file maintains all the requests from the client to the middlewares. The requests are maintained in the following format: date-time, port, server-Id, object name and function name.

This file is used by M_2 to identify and process the recent requests given to M_1. When M_1 is dead for example consider a situation where a client sends a request at time 09:01 which is logged in the client-logs file, received and processed by M_1 for the next second 09:02 when client sends next request it is logged in client logs file as usual but is not processed by M_1 since M_1 is dead at 09:01 and simultaneously M_2 receivers NAK from M_1 and now M_2 reads the log file to process the recent request to M_1 to continue the process.

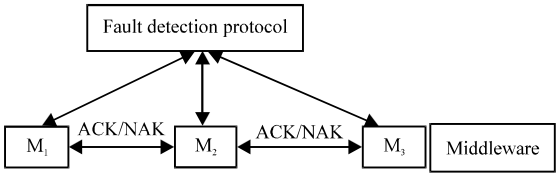

Fig. 2: Fault-free middleware

Data Replication in Middleware (DRM): Data replication is a mechanism of replicating data's such as port-number, object name, function name, server-Id in all neighbor middleware's after processing each request in regular time intervals and also ACK and NAK messages are transmitted and received among neighbor middlewares to know their alive status.

Fault-detection protocol: The mechanism of fault tolerance helps the middleware to continue work even after partial failure. Figure 2 show the design of fault-free middleware. Fault-Detection Protocol (FDP) is used to provide reliable communication between middlewares on the basis of ACK and NAK messages transmitted and received among neighbor middlewares.

The particular middleware is detected faulty, if its fails to response to the are you alive message from the neighbor middleware for M_{MT} time in M_{TO} period. Then, the detected middleware is shunned where it is removed from the cluster and the cluster updates its view to balance the load and client interceptors thereby avoiding the dead node.

Assumptions:

M_{TO}→Middleware maximum time to wait for response
M_{MT}→Middleware maximum number of tries
M_S→Middleware M_i is removed from cluster

```
for each middleware in N_M
  If (MTO == 2000)
    M1 has missed "are-you-alive" message M_MT times
    M1 is shunt and it sends NAK to M2 and dies
  end
end
```

Design of the fault tolerance middleware system: The fault-tolerant middleware mechanism describes the method of choosing the next middleware based on the shortest path between middlewares found using Dijkstra's algorithm and the availability of individual middleware memory and required memory N_{FC} remote function calls of the objects. Figure 3 shows the design of the fault tolerance middleware system where a client requests an object from server in which the request is logged in log file and from their request is forwarded to the server through a middleware. The main objective of the middleware is to check the load balance of various servers and to find the appropriate one which is able to satisfy the

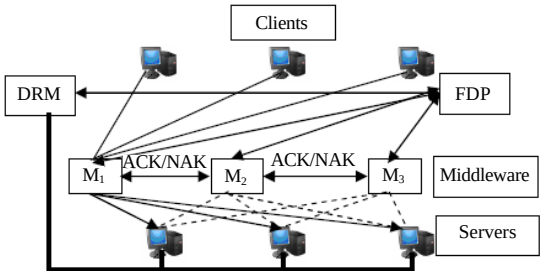

Fig. 3: Fault tolerance middleware system

request of the client. When a middleware is identified as fault based on fault detection protocol which monitors all the middlewares then the process is taken over by the next neighbor middleware that is identified based on the fault tolerance protocol.

Assumptions:

N_M→Number of available middlewares
S_{AM}→Size of available memory in a middleware
A_{RM}→Size of required memory by the N_{FC} remote function calls of the objects.
Md_{ij}→Weight of edges between i and j middlewares. $MD_{ij}^{(m)}$→Weight of minimum cost path between i and j middlewares.

```
for each middleware in N_M
// shortest path using Dijkstra's algorithm
MD_ij^(0) 0 for all i, Md_ij^(0) = ∞, for i≠j;
 For m = 0 to N-1
    {
   MD_ij^(m+1) = 0, for all i;
    for i≠j;
 MD_ij^(m+1) = min_k∈ij {MD_jk^(m) + Md_ki} for i ≠ j;

}
 end
    If (Middleware1 == failure)
          for each middleware in N_M
              IF (S_AM > S_RM)
              Call next Suitable M2
               end
        end
   end
  end
```

EXPERIMENTAL RESULTS

Researchers implemented the proposed approach in Java to test the efficiency. Researchers had some servers, middlewares and clients. The memory consumed by the objects and the time consumed by every method calls were stored in the databases.

All the remote calls of methods were monitored and information like called object, method, total time, elapsed time and memory occupied were also stored in the database. Figure 4 shows the one of the clients request to middleware through the log file. Figure 5 shows the process of current middleware in which the following details are displayed:

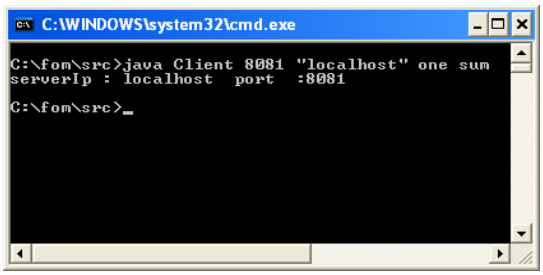

Fig. 4: Client's request to middleware

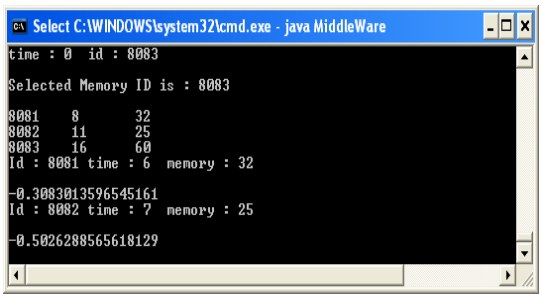

Fig. 5: Current middleware's process

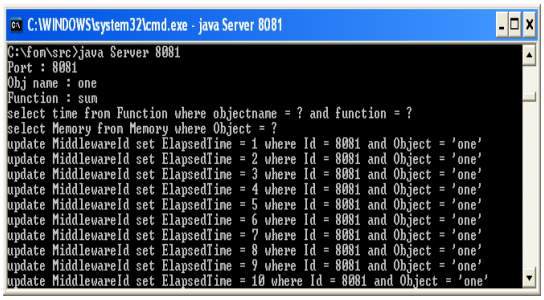

Fig. 6: Server's operation

- Time taken by the server to complete the current task and the server Id
- Based on above said details the server (8083) is selected
- Continuous update of server details such as Id, time consumption and memory capacity
- CPU usage value that is being updated for every 1000 msec. Figure 6 shows the performance of the current server's operations based on the input from the middleware

CONCLUSION

Probing the vital parameters of a middleware such as fault tolerance, response time and replication and monitoring before designating it for handling a request proves worthwhile. Significant are the performance improvement and fault tolerance ratio in experimental study. Considering the parameters that identifies the fastest/fittest among a group of middleware ensures optimal resource utilization and helps speedy completion of any task. Researchers have also handled the replication issues on the middleware side. With proven experimental results, the importance of appraising a middleware before delegating it for a task becomes indispensable.

ACKNOWLEDGEMENT

The comments from the reviewers have added quality to this study and their research ought it be received with words of appreciation.

REFERENCES

Armendariz, J.E., H. Decker and F.D. Munoz-Escoi, 2006. Boosting the availability of information system by data replication. http://web.iti.upv.es/~armendariz/pubs/pdf/2006/CAiSE_poster.a1.pdf.

Dumitras, T. and P. Narasimhan, 2007. Got predictability?: Experiences with fault-tolerant middleware. Proceedings of the 2007 ACM/IFIP/USENIX International Conference on Middleware Companion, November 26-30, 2007, Newport Beach, CA.

Goel, S. and R. Buyya, 2006. Data Replication Strategies in Wide Area Distributed Systems. In: Enterprise Service Computing: From Concept to Deployment, Qiu, R.G. (Ed.). Idea Group Inc., Hershey, PA, USA, pp: 211-241.

Gokhale, A.S., B. Natarajan, K. Joseph, C., Douglas and C. Schmidt, 2004. Towards real-time fault-tolerant CORBA middleware. Cluster Comput. J., 7: 331-346.

Kim, K.H., 2001. Middleware of real-time object based fault tolerant distributed computing systems: Issues and some approaches. Proceedings of the 2001 Pacific Rim International Symposium on Dependable Computing, December 17-19, 2001, Seoul, Korea, pp: 3-8.

Kim, Y., E.K. Kim, B.J. Jeon, I.Y. Ko and S.Y. Park, 2006. Wapee: A fault-tolerant semantic middleware in ubiquitous computing environments. Proceedings of hte EUC 2006 Workshops: NCUS, SecUbiq, USN, TRUST, ESO and MSA, August 1-4, 2006, Seoul, Korea, pp: 173-182.

Marculescu, D., N.H. Zamora, P. Stanley-Marbell and R. Marc-Culescu, 2003. Fault-tolerant techniques for ambient intelligent distributed systems. Proceedings of the International Conference on Computer Aided Design (ICCAD'03), San Jose, CA.

Murray, P.T., R.A. Fleming, P.D. Harry and H.P. Laboratories, 1998. Somersault: Enabling Fault-Tolerant Distirbuted Software Systems. Hewlett Packard Laboratories, USA, pp: 98-81.

Natarajan, B., A. Gokhale, S. Yajnik and D.C. Schmidt, 2000. Applying patterns to improve the performance of fault tolerant CORBA. Proceedings of the 7th international Conference on High Performance Computing, ACM/IEEE, December 17-20, 2000, Bangalore, India, pp: 107-120.

Poza, J.L., J.L. Posadas and J.E. Simo, 2009. QoS-Based Middleware Architecture for Distributed Control Systems. Springer, Berlin Heidelberg, New York, pp: 587-595.

Schantz, R.E. and D.C. Schmidt, 2002. Research advances in middleware for distributed systems. Proceedings of the IFIP 17th World Computer Congress-TC6 Stream on Communication Systems: The State of the Art, August 25-30, 2002, Montreal, Canada, pp: 1-36.