2011 International Conference on Computer and Software Modeling
IPCSIT vol.14 (2011) © (2011) IACSIT Press, Singapore

DASM: A New Framework for Modeling and Evaluation of Distributed Algorithms

Vahid Khalilpour, Moharram Challenger, Ali Farzan and Vahid Heydarinezhad

Islamic Azad University Shabestar Branch

Abstract. Distributed systems are used widely in the world nowadays and distributed algorithms are among important parts of distributed systems. To run a distributed algorithm, programmer should be involved with network programming details more than details of distributed algorithm itself. Also, studying the behaviors of distributed algorithms during its execution and statistic results after execution is very important to distributed programming. In this paper, a new simple platform, called DASM, is introduced which helps programmer to implement and evaluate any distributed algorithm. Programmer can implement his/her distributed algorithm using DASM primitives, and monitors the results in diagrams during execution. User can make any kind of message and exchange them between processes.

Keywords: Distributed Algorithm, Modelling, Visualization, Evaluation Platform.

1. Introduction

Distributed systems are important and open to do more researches. But as a principle writing a distributed program is more difficult than centralized one [1]. Some of the reasons are network based programming, process synchronization like mutual exclusion, fault tolerance, debugging and so on. There are different tools, platforms and frameworks which are introduced in different papers [1][2]. Also comparing distributing algorithms' functionality, correctness and performance evaluations are cumbersome and time consumer [3].In this paper a new platform called DASM[1] will be presented. DASM can model a distributed algorithm's running over a computers network. This new environment can aggregate and present different statistics about messages exchanged between machines during algorithm execution. Distributed algorithm designer using DASM can model his or her algorithm via comprehensive primitives offered by DASM without dealing with details of distributed systems or networking issues.

2. Background

Due to the complexity of building distributed system, various tools have been introduced to aid designers. General purpose message passing tools[1][2][4] present simple services to do message passing. Meanwhile single-purpose systems like TCGMSG [5] are designed to special application. Most of these tools and programming environments presents various computing models like data parallelism, task parallelism and shared memory [3]. Other tools help testing and debugging distributed systems. For example Wang [6] presented a declarative environment which can simulate a distributed system. The result of system execution is presented in the form of reports at the end of execution. However, this tool can only present limited information about number and type of exchanged messages from client to server or vice versa. Another tool called Net Logger [7] can find out the number of messages exchanged in network and the rate of processor's performance. Also it can simply provide statistics on distributed system's performance and network traffic. But it cannot display the state of system, nodes and resources while program is executing.

[1] Distributed Algorithm Simulating and Modeling

ConcurrentMentor [2] is another tool which can model a distributed system in 4 diagrams. This tool is suitable for educational purposes. Students can learn about distributed systems and its algorithms. Hartley presented a visualization technique in which some information about program execution is stored in text file format. XTANGo [8] is used to display the execution after accomplishing programs run. One of the most powerful systems to model distributed systems is PARADE [9] which is built based on POLKA [9]. Systems like XTANGO and PARADE are working in post-mortem way. Its advantage is the utility of stored data in any time to animate and visualize the execution. Its disadvantage is that the stored data needs a huge space and the execution should be repeated completely in the case of any change in program. Another tool called DisASTer [10] is a platform to run distributed algorithms. This tool presents different diagrams for running processes. It simulate algorithm's execution using threads on a computer and so it has limitation in number of process.

3. Distributed Algorithm specification

Algorithms for distributed systems are completely different from sequential algorithms [3]. Delay in the networks is very bigger than processor's run time. Therefore, the main goal in distributed algorithms' design is reducing the number of messages that should be communicated. Fortunately, to design a distributed algorithm for an application, we mostly need a set of classic repeated tasks [3]. For example send, receive, broadcast, semi-cast, gather, synchronization and so on. Many other real algorithms like election, mutual exclusion, fault-detection, taking snapshot, determination detection, and fault-tolerance are implemented using these primitives [3]. However, some algorithms organize processes in centralized manner [11] and some other in distributed way. Centralized approaches are simple but with less performance rather than distributed ones. DASM supports both of them. It is supposed that all of the processes can communicate with each other. Each process can have its own predefined information, like name, ID, neighbours' ID and so on. This tool can present a full report from messages during execution and after execution. Message type can be defined by user. It can be different for any algorithm to satisfy various needs such as causality ,request , data exchange and so on.

4. System Model

DASM is designed very simple which a user can inherit a few classes and implement his or her algorithm. Running this algorithm he can observe different views of algorithm. After implementing the algorithm, user can define the location of each process and run the algorithm. If the location was not predefined, this tool locates them according to load of each machine. The tool can be run in the real network or a single computer. In network, processes are dispenses over machines, but in a single computer it simulates processes using threads. it is implemented in Java and has 3 main components:

DASM Toolkit: DASM has a management console and a server to detect and connect to other computers. This console has windows to show processes states, message exchange, and network agents. Each computer running Agent connects and registers to DASMServer which is in DASMConsole file. Running DASMserver, detected Agents' list windows is shown. DASMServer's main duty is interacting with user, displaying current state and managing Agents. DASMServer loads the processes according to user's request or its own load balancing strategy.

Agent: Agent package has a few communicating classes and an executable class called Agent. This package should be in any client computer participating in distributed algorithm. After running Agent, it connects to DASMServer and is ready to execute any submitted process. Algorithm processes are made from TaskManager class and sent to Agents. An agent can run several processes using separate threads. Each agent has 2 channels to communicate with other processes which are called MessageQueue to save arriving messages and MessageDispatcher to send a message.

View: This package includes all of the classes to display the state of running processes in the system namely, TopologyPanel, SequentcePanel, ProcessPanel and MessageRatePanel which any of them displays different view of algorithm in a graphical manner. DASM collaboration diagram for main components are shown in figure 1.

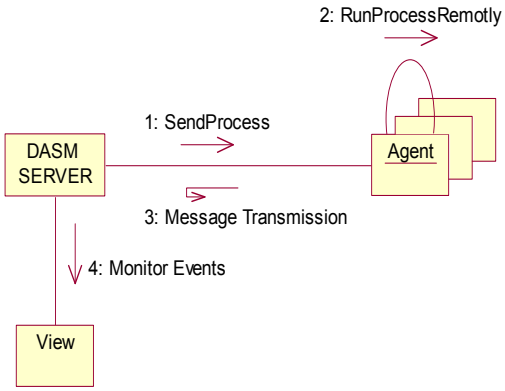

Figure 1: DASM collaboration diagram

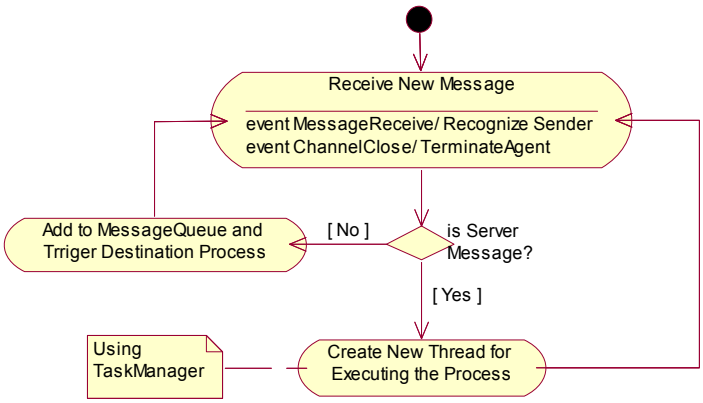

Figure 2: message reception in an Agent

4.1. Network Agent Components

After running DASMServer which gets port number, Agent can run and register itself on server. Then agent waits to receive a message. Receiving TaskMessage means a new process is sent to agent's machine. Agent extracts the process and runs it. To run a process, agent gives the process tow object of communications, MessageQueue and MessageDispatcher, to receive and send message respectively. It is possible that a process sends a message to another process. At this case, receiver agent gets the message and delivers it to determined process in it. According to the type of message, receiver agent will reply as acknowledge. Also according to the type of message, if it is broadcast, the agent delivers the message to all of processes in it. In figure 2 the activity diagram for receiving a message is shown.

Each process in a machine to receive a message can call one of two methods in MessageQueue: synchReceive and asyncReceive. Each process which is derived from PROCESS includes different send and reveive methods like: syncSend, syncReceive, asyncSend, asyncReceive, multiCast, broadCast, semiCast and so on. User can call any method according to his need. For communicating 2 processes using send primitives, message is send from the local agent to DASMServer and DASMServer sends the message to destination. In figure 3 collaboration diagram of process communication is shown.

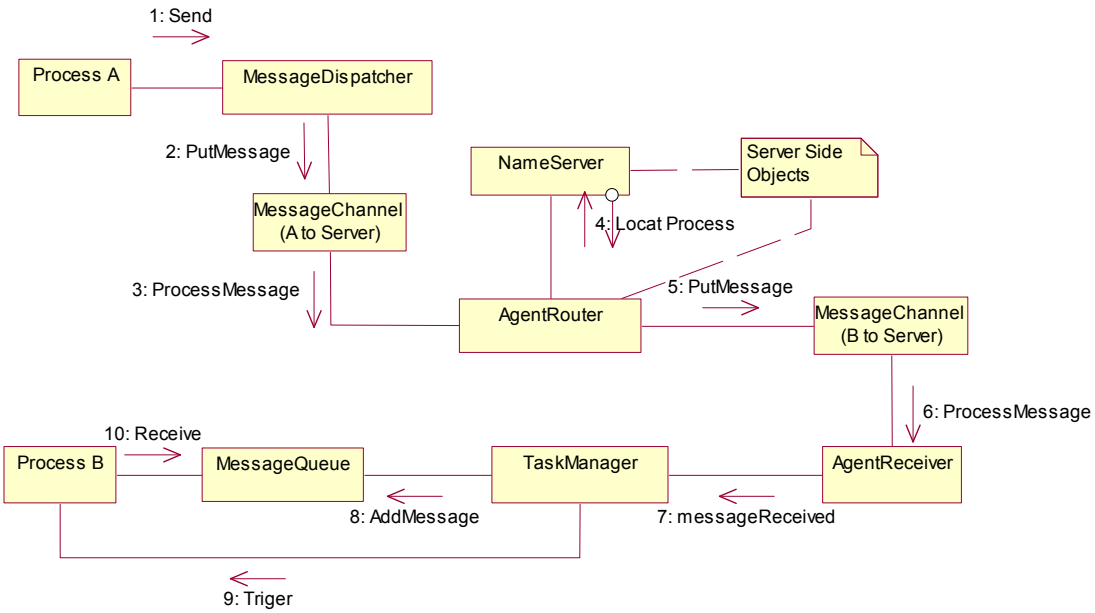

Figure 3: Collaboration diagram for communicating two processes

Sending a message is done using MessageDispatcher via channel between agent and server. The server has a thread of AgentRouter class for each of running agents. It is waiting to receive a message and react to that. If a connection of an Agent is broken, the processes on it will be removed and the agent will be closed.

4.2. DASMToolkit Component

DASMServer package includes classes to interact with user, managing other processes and message routing. Its executable class is DasmConsole which makes a window to user to show the list of agents. Also it displays message transmission during algorithm execution.One of the important parts of DASMServer is NameServer which holds agents information in it. It helps to route the messages and View component displays this communication. In figure 4, the collaboration diagram for registration of an Agent is shown. After registration of an agent in NameServer the View is refreshed and new agent is shown in window.

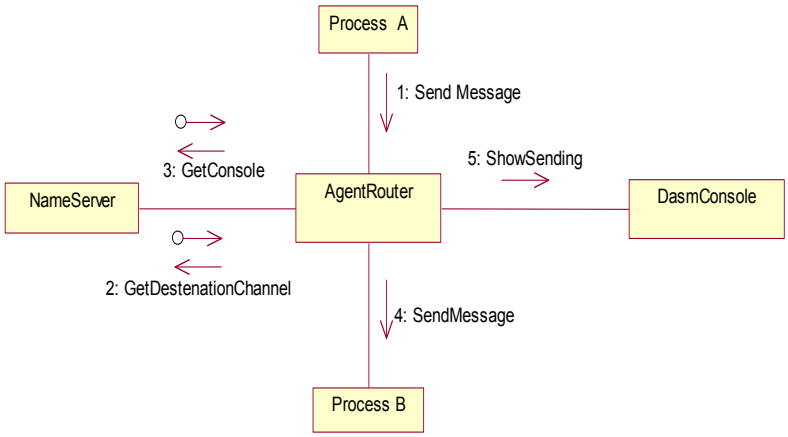

Figure 4: Collaboration diagram for registration of an Agent

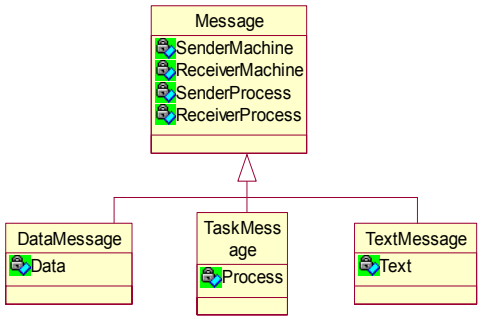

Figure 5: Message inheritance

4.3. NetworkView Component

This component is responsible to display different events during algorithm execution. This component is informed from message exchanges using DASMConsole and displays the communications in graphical window. The component has different views. TopologyView shows system agents and displays animated token in case of communication between processes of agent. SequenceView shows causal sequence of send and receive between processes. ProcessView shows communications between processes. MessageTrafficView shows the amount of message submission for any process in the form of bar-diagram.

4.4. Messages

Communicating is done using different messages. To build various message user can use DataMessage. Any kind data can be sent in a message. All of the messages in DASM should be inherited from Message class (figure 5). Each message has public information like sender machine address, sender process name, receiver machine address and receiver process name. The receiver machine address is not compulsory and DasmServer can find using Naming system. But sender and receiver process name is obligation. Broadcast field in a message can be set, to send it to all running process.

5. Modelling an Algorithm Using DASM

DASM is network-topology free and DASMServer can run over any arbitrary network. Running DASMServer makes first agent itself. All agents are shown in DasmConsole (main windows) with IP. Any computer can run one agent, but an agent can run many processes in it. DASMConsole and Agents should get two command prompt arguments before running. First their machine name, second port number. There is a list of “Recognized Agents” in the right part of main window showing IP of running agents in system.By right clicking on any of the machines in this list, you can send a text message to that agent or disconnect it.

5.1. Implementing an algorithm in DASM

Implementing an algorithm in DASM is simple. You should just inherit the PROCESS class in your algorithm. Then your algorithm will have primitives and able to connect to others with channels (via MessageQueue and MessageDispatcher classes). To communicate with other poetesses, algorithm should build a message. To build a message a new class (containing any data or methods) should inherit from Message class. To run the algorithm the DASMStart.java should be modified. In this class, there is a method called RunProcess which can send process on any agent and run that.

5.2. Case Study: Producer Consumer Algorithm

Producer-consumer algorithm is one the famous concurrency and process synchronization problems. In this section the problem is implemented in distributed form between two computers. The communicating data is considered a simple integer in IntMessage class inheriting Message class.

```
public class IntMessage extends Message {
   private int data;
   public IntMessage(int  o){   data=o;   }
   public int setData(int  o){   data=o;   }
   public int getData(){   return(data);   }
}
```

All of the message information like sender process name sender machine address, receiver process name receiver machine address and so on are inherited from Message class to IntMessage class. The main algorithm is implemented inheriting from PROCESS class.

```
public class Producer extends PROCESS {
   public void Start(){
      int item=0;
      IntMessage message;
      while(true){
        System.out.println(item+"  produced");
        message=new IntMessage(item++);
        message.setSenderProcess(getName());
        message.setReceiverProcess("Consumer");
        AsyncSend(message);
        SyncReceive();
   } } }
```

```
public class Consumer extends PROCESS {
  public void Start(){
  int item=0;
  IntMessage message;
  while (true){
    message=(IntMessage)SyncReceive();
  System.out.println(message.getData()+"Consumed");
     message.setSenderProcess(this.getName());
      message.setReceiverProcess("Producer");
      delay(1000);
      AsyncSend(message);
  }}
```

The Producer class is inherited all of its required communicating methods, among them an abstract class called Start. Derived class should override the Start method and implements algorithm in it. First line, the setName method assigns process name in desired name. If it is not set, class name is chosen as its name. Second line makes a message from algorithm message class, IntMessage, in which new messages will be held. Next lines are "for loop" including: instantiating a new message from IntMessage class setting the message sender and receiver and finally sending message in asynchronous form and wait for receiving the answer in synchronous form. Consumer process consumes the receiving data and returns it back to producer to ensure producer from consume accomplishment and producing next data.

Like Producer, Consumer is derived from PROCESS, overrides Start method and implements its algorithm in it. It has a loop, waiting to receive a message (synchReceive), consuming the massage's data (printing its number), changing its sender (itself), changing its receiver (Producer) and finally sending it in the asynchronous form after a while (delay to control the interactivity). After implementing Producer and Consumer separately, it's time to run them with changing the DASMStart.java.

```
public class DASMStart extends DASM{
   public void START(){   RunProcess(new Consumer());       RunProcess(new Producer()); }
}
```

START makes any process construction and runs them with RunProcess. In this example it instantiates two objects from Consumer and Producer classes and pass them to RunProcess as parameters. It is possible to add the location where process should run into runProcess's parameters. After algorithm run, we can observe deferent views (figure 6). TopologyView displays the TaskMessages in red and other messages in yellow. It is possible to change the message icon speed in screen from Setting menu. Sequence Diagram displays the order in which messages are sent between running processes (figure 7).

DASM can report statistics for different aspects of algorithm. Programmer can use these statistics to analyze message communication rate and algorithm fault-tolerance. To have real state in algorithm, programmer can define message and machine lost rate separately in Setting menu. After program run in DASM, programmer can select Statistics menu to view number of processes, exchanged messages, lost messages and system's overall status.

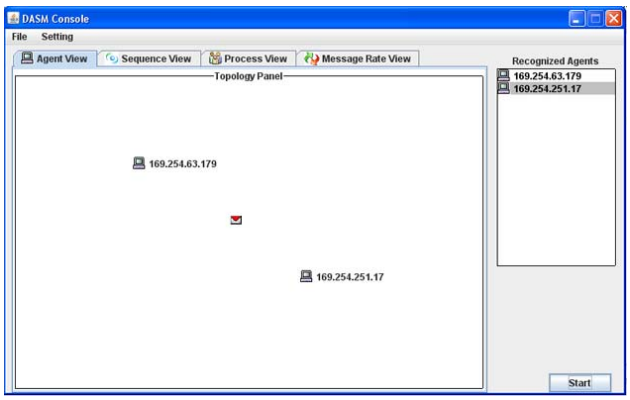

Figure 6: Topology View

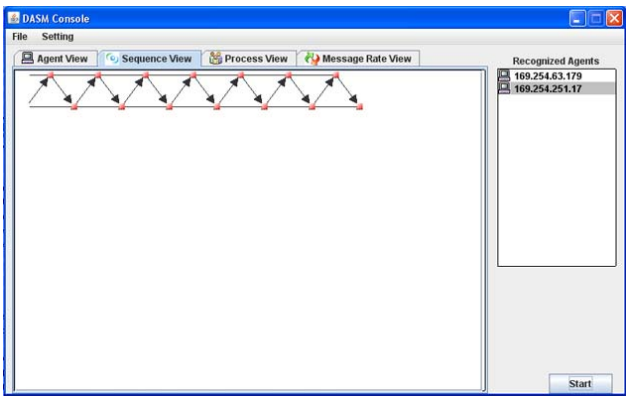

Figure 7: Sequence Diagram

6. Conclusions

In this paper a new environment to modeling and monitoring distributed algorithms execution is presented. One of the significant features of the new framework is that algorithm really runs over a network. Also it is very simple to use. Programmer can use the framework to implements his distributed algorithm with inheriting some classes. He calls required Message-passing based primitives inherited from main classes to avoid dealing with network communication details. Running such implemented program, he can monitor different views of algorithm's execution like topology view, communication view, messaging sequence and so on. Also he can display statistics like message number altogether or individually for any process/machine/agent, message lost, crash rate, traffic rate and so on. User can define required message types in algorithm inheriting from Message class and send its instantiates from any process to another one. Finally user can define his desired rate of crash or message lost and evaluates algorithm's fault-tolerance.

7. References

[1] T. Fahringer, A. Jugravu. Java Symphony: New Directives to Control and Synchronize Locality, Parallelism, and Load Balancing for Cluster and GRID-Computing. *Proc. ISCOPE 2002*. Washington. 2002: 3-11.

[2] S. Carr, C. Fang, T. Jozwowski, J. Mayo and C. Kuang. ConcurrentMentor:A Visualization System for Distributed Programming Education. Proc. *International Conference on Parallel and Distributed Processing Techniques and Applications*. Las Vegas, Nevada, USA, 2003: 6-23.

[3] N. Santaro. *Design and Analysis of Distributed Algorithms.* Publication place: JohnWiley&Sons. Hoboken, New Jersey, 2007.

[4] S.Parsa, V.khalilpour .Automatic Distribution of sequential Code using JavaSymphony Middleware. *Lecture Notes in Computer Science*. 2006. 3831*:* 440-450.

[5] G. Mattson. Programming Environments for Parallel and Distributed Computing. *International Journal of High Performance Computing Applications.* 1995. 9(2): 138-161.

[6] Y. Wang, M. J. Rutherford. Automating Experimentation on Distributed Testbeds. *Proc. 20th IEEE/ACM International Conference on Automated Software Engineering.* Long Beach, California, USA. 2005: 7-11.

[7] B. Tierney, D. Gunte. NetLogger: A Toolkit for Distributed System Performance Tuning and Debugging. *LBNL Tech Report.* LBNL-51276, 2003.

[8] J. Hartley. Animating operating systems algorithms with XTANGO. *Proc. 25th SIGCSE technical symposium on Computer science education*. New York. 1994: 6-14.

[9] J. T. Stasko. The PARADE Environment for Visualizing Parallel Program Executions: A Progress Report. *Technical Report GIT-GVU-95-03*. Georgia Institute of Technology. Atlanta. January 1995.

[10] R. Oechsle, T. Gottwald. DisASTer (Distributed Algorithms Simulation Terrain): A Platform for the Implementation of Distributed Algorithms. *Proc.10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE2005.* Caparica, Portugal, 2005: 6-27.

[11] M. Challenger, V. Khalilpour, P. Bayat, M. R. Meibodi. A New Robust Centeralized DMX Algorithm. *Proc. IASTED PDCN2007.* Innsbruck, Austria, 2007: 2-13.