MODELLING A RELIABLE DISTRIBUTED SYSTEM BASED ON THE MANAGEMENT OF REPLICATION PROCESSES

Cezar TOADER
Technical University of Cluj-Napoca, Romania, cezar.toader@cunbm.utcluj.ro

Diana Cezara TOADER
Bucharest University of Economic Studies, Faculty of Management, Romania, diana.cezara@gmail.com

ABSTRACT
In the modern economy, the benefits of Web services are significant because they facilitates the automation of activities in Internet distributed businesses and also the cooperation between organizations. The interconnection of processes running in different computer systems is now possible by using Web services. This paper presents a model for a reliable distributed system and its development stages. This paper describes the communication between the system processes, based on the message exchange, and also presents the distributed agreement among processes. For the fault-tolerant systems a list of objectives is defined and, further, a framework model for distributed systems is proposed. This system framework makes distinction between management operations and execution operations. The proposed model promotes the use of a central process especially designed for the coordination and control of other processes. This model of a reliable distributed system could be used as a platform for many models of distributed systems, based on the management of replication processes.

KEYWORDS: *distributed systems, reliability, replication management, information systems.*

JEL CLASSIFICATION: *C61, C88, Q55*

1. INTRODUCTION

The development stages of a model of a distributed system is presented in this paper. The primordial abstract elements are: the process and the connection between processes. The problem of communication between the processes within the distributed system is formulated, the concept of message exchange between processes is used, and also the problem of distributed agreement among processes in presented here.
This paper will define a model of a distributed system in which some processes have the role of management of the operations carried out by other processes which have the execution role. After presenting of the processes of the system components, there is an abstraction of the exchange of messages between the processes of the distributed system. Further, an abstract model of a distributed system is proposed and analysed, which contains worker-processes controlled by a manager-process, all of them running on different hosts. The system model described below could be considered as the basis for the development of a distributed system tolerant of faults, which uses the replication of operations.
On top of the abstract system model previously presented, a replication model is proposed, targeting operations and data within a distributed system containing several worker-processes. After the presentation of a list of objectives for the fault-tolerant system, a logical separation of management operations from the execution operations and data storage is proposed.
Web services are increasingly used in distributed systems, applications and services of outmost importance. For distributed systems, the architecture oriented on services (SOA - Service Oriented Architecture) is already accepted as being the architecture able to interconnect

applications running on different systems and facilitates the complex interactions between autonomous systems and heterogeneous ones, either within organizations or between organizations in relations of Business-to-Business type.

2. THE AVAILABILITY AND RELIABILITY OF A DISTRIBUTED SYSTEM

The failures and damages of web applications may lead to incorrect processing or even to the system failure in the business of e-commerce type, e-banking or other systems based on transactions. One of the important causes of the services' interruptions is the so-called „server falling". Therefore, there are becoming increasingly important the techniques which provide fault tolerance and the provision of the services on Internet even in the conditions of damages to the servers.

For many distributed systems based on Web services there are a large number of client applications already installed and is very complicated the modification of all of these applications. This is the reason why the service providers search for specific fault tolerant solutions, called client-transparent, which do not require any special action on the part of the client applications or any amendments of them. This transparency on the part of client application is an important requirement for both the application itself, as well as for the operating system in question.

In complex applications, Web services should be connected to other services with a view to form composite Web Services and complex architectures based on services. If a component in the chain of services is not available or is not reliable, then the entire system is affected.

A correct service is the one which implements the function of the system concerned. When the service provided by the customer differs from the correct service, it means that there is a failure of the system. This deviation means that the service does not comply with its well-defined specifications. An error is that part of the system status which may cause a failure. The cause of the error is a fault. A fault can be active or not. When it is active, the fault produces an error in the system. Serious errors lead to the system crash (Avizienis et al., 2004).

The system *availability* refers to the ability of a system "to be ready to provide a service correctly". The system *reliability* refers to the ability of a system "to be able to continuously provide a correct service". *Dependability* is a more comprehensive concept which incorporates several components: availability, trust, security, confidentiality, integrity, maintainability (Avizienis et al., 2004).

A very important means to achieve the reliability is the fault tolerance. This refers to the techniques meant to give the system the ability "to provide a service correctly even in the presence of errors". In the systems considered reliable, data replication is a technique that has been widely accepted that allows the avoidance of system crashes. Thus, the architects of the system implement a service within a distributed system using a group of servers, independent from a physical point of view in such a way that if a part of them cease to operate, the remaining servers have the ability to provide the service in question to the clients.

Replication protects an application running on a server against the faults, so that if a replica becomes inoperative, other replica is available to provide that service to customers. The most frequently used replication strategies are classified as follows: liabilities, active and semi-active. An overview of these strategies can be found in (Moser et al., 2007).

2.1. Traditional replication techniques and their limitations

Data replication consists of maintaining several copies of data, called replicas, on separated computers. Replication is an important technology in the field of distributed services.

Replication improves the availability of the data by means of enabling the access of users to duplicate data in the vicinity, even when some copies of the requested data are not accessible.
Traditional techniques of replication are aimed at maintaining the consistence between the main package of data and a single copy, and the client applications could "see" a single set of data with high degree of availability. The basic concept is as follows: the access to replicated data is locked until they are up-to-date, and this update is proven within the application. This is the reason why such techniques are called "pessimistic".
The replicated entity could be of several types: data object, file, data structure, and service. There are at least two categories of replication techniques:

- Techniques targeting the replication of databases;
- Techniques targeting the replication of objects and processes in a distributed system.

The techniques of the two categories above have many similarities, but also important differences (Wiesmann et al., 2000). Due to the continuous progress of Internet technologies, it appears the tendency to apply the algorithms specific to pessimistic replication to wide area networks (WAN).
However, in this case, there are not expected the same performance and the same data availability as in the case of local area networks because of the reasons mentioned in the following paragraphs.
Firstly, the Internet is relatively slow and does not provide the same reliability and the same data availability as local networks (Dahlin et al., 2003).
Secondly, the algorithms of pessimistic replication are not easily to scale for being used in WANs. It is difficult to build on a WAN a pessimistic replication system, having a large size, in which frequent updates of data appear because the increasing number of served sites leads to the degradation of important performance parameters: response times and availability of data (Yu & Vahdat, 2006). That is the reason why many Internet services use techniques for *optimistic* replication.
Thirdly, certain activities of the human users require data-sharing. In many engineering areas and especially in the software engineering, the specialists often work on punctual problems, well delimited and specified until the completion of the work in question. Therefore, it is better to allow independent updates of data in centralized warehouses and subsequently to solve conflicts, instead of blocking the access to a specific set of data until a certain human operator finishes editing (Vesperman, 2006).

2.2. Optimistic versus pessimistic replication

Optimistic replication signifies a set of techniques for the efficient sharing of data in mobile or large size working environments. The essential feature that distinguishes the algorithms for the optimistic replication from those of pessimistic replication is the manner in which the control of concurrent access is approached.
The pessimistic algorithms coordinate the synchronisation between replicates during access and block the new requests of users for the duration of an update in progress. The optimistic algorithms allow access to data even if it has still not been completed the synchronization of replicas on the basis of the optimistic hypothesis that conflicts will appear only rarely, or not at all. The updates are sent to replicas in the background, and the occasional conflicts are solved when they occur. This optimistic way of seeing things is not new, but was widely spread as the Internet and mobile technologies have become more and more used.
As compared to the traditional techniques of pessimistic replication, optimistic replication promises a high degree of system performance and of data availability, but accepts for a limited duration some differences between the main data and their duplicates, this inconsistency being solved in due time.

In the end, there has to be said that the optimistic algorithms give a fast response to client applications as they carry out the updates of data as soon as they have been requested.
These advantages require a cost. Any distributed system is faced with the problem of a compromise between the consistent data and their availability (Pedone, 2001). In the early stages of work where the pessimistic algorithms are waiting to achieve a certain status of the system, optimistic algorithms allow the continuation of the operations with the client applications, but in the background, they are carrying out other operations.
The optimistic algorithms have to cope with the situations in which there are replicas different from each other, which may lead to conflicts between concurrent operations. Therefore, they are suitable only to applications that can tolerate occasional conflicts and data inconsistency for a limited time (Saito and Shapiro, 2005).

3. MODELLING THE REPLICATION IN DISTRIBUTED SYSTEMS

Two abstract elements which allow the representation of the physical infrastructure of the system are used: *the process*, and *the connection* between processes.
Within the framework of a distributed program, the process is abstracting an active entity which runs with respect of a certain algorithm and performs a certain processing of specific data. The process may represent a computer, a computer processor or an execution thread.
The processes should be able to cooperate for the fulfilment of common tasks. Therefore, they should exchange messages between them. The messages between processes are possible only if there is a specific physical link between processes. The connection is abstracting the link between processes and lies at the foundation of the communication between the processes.
Organized in a certain way, the links form networks. The communication between processes involves the following components: the message *m*, the sender process *S*, and the receiver process *R*, as seen in Figure 1.

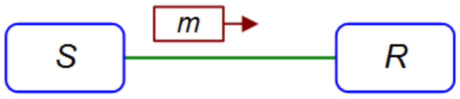

Figure 1. The communication between processes
Source: adapted from Toader (2011)

There can be build systems in which each process has a certain role in the framework of the distributed system, and in order to fulfil the role, the process must be able to carry on specific activities. In these cases, it is important the nature of the messages received and sent between processes. In the case of some complex distributed systems, more processes can be defined, with the role of execution, noted, for example, X1, X2 and so on, located on different hosts, which carry out specific activities after receiving specific message from a process with the role of manager, noted, for example, with M, as seen in Figure 2.

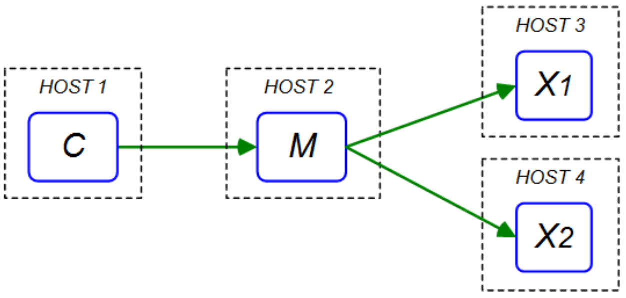

Figure 2. The process M plays the role of manager for the worker-processes X1, X2
Source: adapted from Toader (2011)

Consider now the two processes, S (sender) and R (receiver), and an established connection between them. Over this connection the processes send their messages back and forth.
To model the communication between processes, consider now the point-to-point connection between two processes as a software entity (an object) called *conn*, which is an instance of a class called *PointToPointConnection*.
At the level of the object conn there are taking place two distinct events, one at the end of the process from the source process S and the other one, at the end of the destination process R.
The event which models the sending of the message *m* by the source process *S* is:

$$\{\text{conn, Send} \mid \text{R, m}\} \tag{1}$$

The event which models the receipt of message *m* to the process *R* is:

$$\{\text{conn, Receive} \mid \text{S, m}\} \tag{2}$$

In the relation (1), the attributes of the event are: the recipient *R* and the message *m*. In the relation (2), the attributes of the event are: the transmitter *S* and the message *m*.

In the distributed system model analysed in this paper, the correctness of the communication between processes assumes the compliance with the following *properties of secure communication between the correct processes of the distributed system*:

- P1. *Reliable delivery of messages*. If the process S, which shall be carried out correctly, sends a message m to the process R, which also shall be carried out correctly, then the process R possibly receives the message m.
- P2. *The messages are not duplicated to the source*. A fair process at any stage of its execution, sends a message once.
- P3. *The messages are not created at the recipient*. If a message has reached the destination process R from the sender process S, then the message has surely been sent by the process S before the process R.
- P4. *The messages are received in the strict order of their arrival* (FIFO). If the process R has received from another process S firstly the message m1 and subsequently the message m2, then the correct operation of the process R means taking the messages in order: firstly the message m1, and afterwards the message m2.
- P5. *Compliance with an agreement at the level of the distributed system*. If a message was delivered successfully to a process R that is correct, then the message is eventually delivered successfully to any correct process in the system.

A few possible situations should be highlighted:

- Not all messages sent will be received (network-related problems);
- Not all received messages must be delivered to the processing components of the process (can be incorrect messages or with another recipient).

4. MANAGEMENT AND EXECUTION PROCESSES

4.1. Separation of system processes based on their role

Consider now a distributed system in which there are several processes with well-defined roles, located on several hosts, as seen in Figure 3, below.

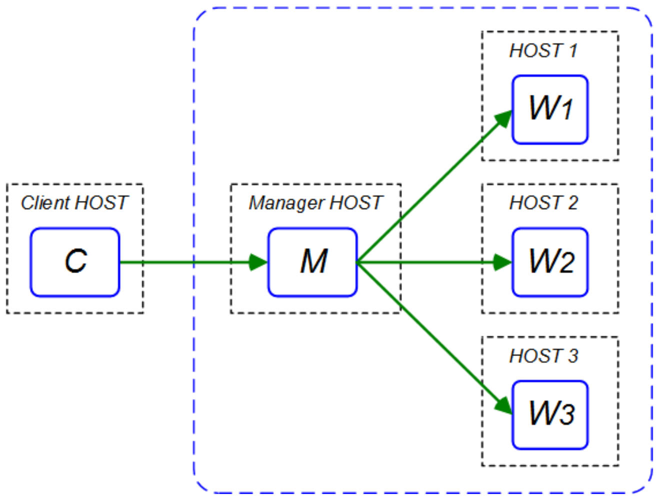

Figure 3. The proposed distributed system model with centralized management and separate execution

The main features of this system are below:

- The process C, referred to as the client-process, has established a secure connection with the process M, referred to as the manager-process.
- The process M is located on another host than the process C.
- Over the secure connection between these processes, there are messages sent from C to M, called *requests*, and also messages from M to C, called *responses*.
- The process M receives the request messages sent by the clients and delivers them to the system components with respect of the receiving order.
- The manager-process M is capable of storing the requests made by the client C in a form which permits subsequent finding of a certain request.
- The system also contains the processes W1, W2, W3, referred to as worker-processes, which have each established a secure connection with the manager-process M, but they do not have direct links between them.
- The worker-processes W1, W2, W3 are located on different hosts, other than the host of process M.
- The worker-processes W1, W2 and W3 are deterministic, in the sense that when they receive identical messages from the process M, they will carry out the same operations and formulate identical responses which they send back to the manager-process M. Therefore, there are no local factors that could determine different responses to identical requests from the manager-process M.
- As a result of receiving requests from the client-processes, the manager-process M sends the messages to the worker-processes W1, W2 and W3, in the same order.
- The content of the messages sent by the manager-process M to the processes W1, W2, W3 is determined by the two factors below:
  1) The content of the request sent by client-process C;
  2) The existence of an agreement on the messages format, for the entire system.
- The manager-process M receives the responses sent by the workers W1, W2, W3, and processes these responses in a separate phase of its execution, in order to form a response message for the client.

The advantages of this system with centralized management and separate execution are:

- The use of the processing power of multiple computers due to the separate physical location of these processor-intensive processes;
- The location of the stored data is closer to the other systems that may require data access through the use of multiple, separately located, worker-processes;
- Less response time of manager-process M in relationship with the client process C through the separation of management-process and worker-processes;
- Ease of worker-processes debugging, based on their separate location and execution;
- The ability to add, relatively quickly, other workers in the system;
- The ability to use worker-processes W2, W3 … Wn, with the role of replicas of a main process W1, and having a processing logic identical to the main process W1, and controlled by the same separate manager-process M.

4.2. The exchange of messages between the client and the manager

As a result of the user actions, the client-process C sends a message to the distributed system, practically, to the manager-process M, requesting the execution of an action and obtaining a response message. This response message contains data and their volume varies depending on many factors, the most visible being the nature of the requested action.
The procedure which determines the send of a message to the process-manager is:

$$\text{Execute } (\textit{action, params, clientID}) \tag{3}$$

This procedure determines the creation of an event at the level of the client-process:

$$\{\, \textit{client}, \text{ Send } \mid [\textit{action, params}], \textit{manager, self} \,\} \tag{4}$$

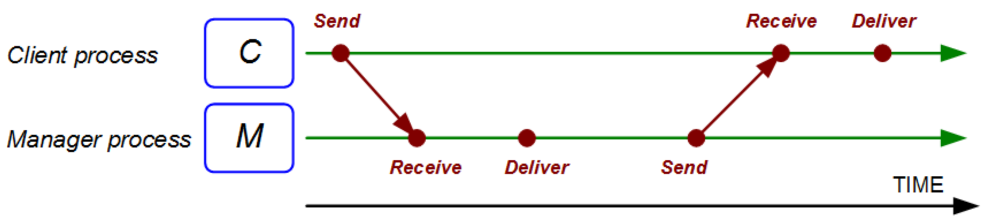

Figure 4. The exchange of messages between the client-process C and the manager-process M

The message reaches the manager-process M and as a result, at the level of M several procedures are successively triggered:

Receive (*action, params, client*)
[*operationID, operationName, parameters*]:= Operation (*action, params*)
Log (*operationID, operationName, parameters*)
Execute (*operationID, operationName, parameters, worker*)
Send (*result, client*)

4.3. The exchange of messages between the manager and the worker

As a result of a request previously made by a client-process, at the level of the manager-process M, after a request registration, the following procedure is executed:

$$\text{Execute (operationID, operationName, parameters, worker, self)} \tag{5}$$

This procedure determines the creation of an event at the level of the manager-process M:

{ *manager*, Send | [*operationID, operationName, parameters*], *worker, self* } (6)

As a result, at the level of the worker-process W, these procedures are successively triggered:

Receive (*operationID, operationName, parameters, manager*)
Log (*operationID, operationName, parameters*)
Execute (*operationName, parameters*)
Send (*operationID, result, manager, self*)

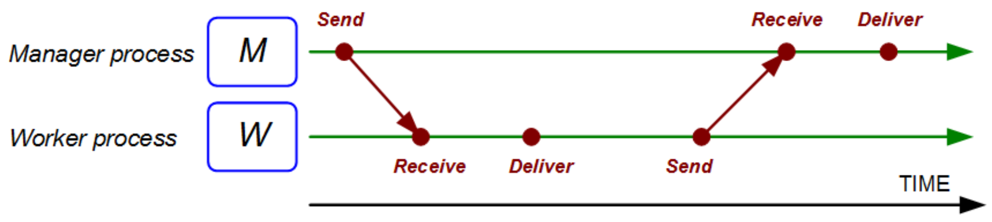

Figure 5. The exchange of messages between the manager-process M and the worker-process W

The procedures mentioned above use the type of data called *operation*. Actually, it is a public class in the system. Therefore, within the system, there is an agreement for the use of this class, referring to the transport of data between the software components.

5. CONCLUSIONS

The distributed system proposed in this paper has been analysed on the basis of primordial elements: the process, the connection between processes and the message.

Within the process model, the software components and the process levels were abstracted. Subsequently, the communication between the process components based on events has been modelled. There have been analysed some problems which influence the functioning of a distributed system as a whole, namely, the matter of distributed agreement and the matter of causal order of messages.

A specific distributed system model was proposed in this paper, which contains several separate processes having the role of workers and the related management execution operations are centralized in a separate process of management. Thus, the specific concerns related to these two categories of operations, management and execution, were separated.

The management and the distinct development of the two process categories is considered to be an advantage brought by the proposed system model. Moreover, the separation of management and execution brings new benefits coming from the use of the processing power on multiple computers.

In the model proposed in this paper, the manner of abstraction the communications between processes, based on messages, has been presented. Subsequently, the abstraction of communications between the client and the manager, and also the communications between the manager and workers was described in details. This model of a reliable system could be a foundation for an entire class of distributed systems models based on the management of process replication.

REFERENCES

Avizienis, A., Laprie, J.C., Randell, B. & Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing, *IEEE Transactions on Dependable and Secure Computing*, vol.1, no.1, pp.11-33.

Moser, L.E., Melliar-Smith, P.M., and Zhao, W. (2007). Building dependable and Secure Web Services, *Journal of Software,* vol.2, no.1, pp.14-26.

Pedone, F. (2001). Boosting system performance with optimistic distributed protocols, *IEEE Computer*, Vol. 34, Issue 7, USA, pp.80–86.

Saito, Y. & Shapiro, M. (2005). Optimistic replication, *ACM Computing Surveys,* vol.37 (1), pp.42-81.

Toader, C. (2010). Increasing Reliability of Web Services, *Journal of Control Engineering and Applied Informatics, vol.12, no.4,* 30-35.

Toader, C. (2011). *Contributions to the architecture and dependability of Web service-oriented distributed information systems,* Timișoara, Romania.

Vesperman, J. (2006). *Essential CVS,* 2nd Edition, O'Reilly Media Inc., USA.

Wiesmann, M., Pedone, F., Schiper, A., Kemme, B. & Alonso, G. (2000). Understanding replication in databases and distributed systems, *Proceedings of 20th International Conference on Distributed Computing Systems (ICDCS'2000),* Taipei, Taiwan, ROC, pp.464-474.

Yu, H. & Vahdat, A. (2006). The costs and limits of availability for replicated services. *Journal of ACM Transactions on Computer Systems,* Vol. 24, Issue 1, NY, USA, pp.70-113.