A Framework for the Engineering of Reliable Distributed Systems

(1) ChandraBose A
Research Scholar,
Madurai Kamaraj University,Madurai.

(2) Dr.K.Alagarsamy,
Associate Professor
Madurai Kamaraj University,Madurai.

Abstract

Engineering distributed systems is a challenging activity. This is partly due to their intrinsic complexity, and partly due to the practical obstacles that developers face when evaluating and adjusting their design and implementation decisions. This paper aims to design framework to automate experiments. Keeping all facts, experiment automation framework is designed in a generic and programmable way to be used with different types of distributed systems for wide-ranging experimental goals. The models are used by generative techniques to automate construction of a control system for deploying, executing, and post-processing the specific experiment. We have validated our approaches by performing experiments with a variety of distributed systems on different test beds to achieve wide-ranging experimental goals.

Keywords: Generative Programming, Simulation, Distributed systems, Framework

1. Introduction

The vision of this paper is to support systematic, repeated experimentation with distributed systems, especially highly distributed systems, in realistic network environments by creating a consistent, unified, reliable experimentation framework. For any experiment with a distributed system, this framework helps software engineers quickly setup each trial control system that is customized to the specific trial, and easily manage the execution and analysis of the trial. Through it, software engineers can achieve efficient cost savings in conducting experiments by automating their experiment trials. The workloads for different experiment trials are to be produced by different numbers of user actors using the same behavior model just described. We first programmed the actor behavior model. Then for each different trial, we simply adjusted the number of user actor declarations in the actor configuration.

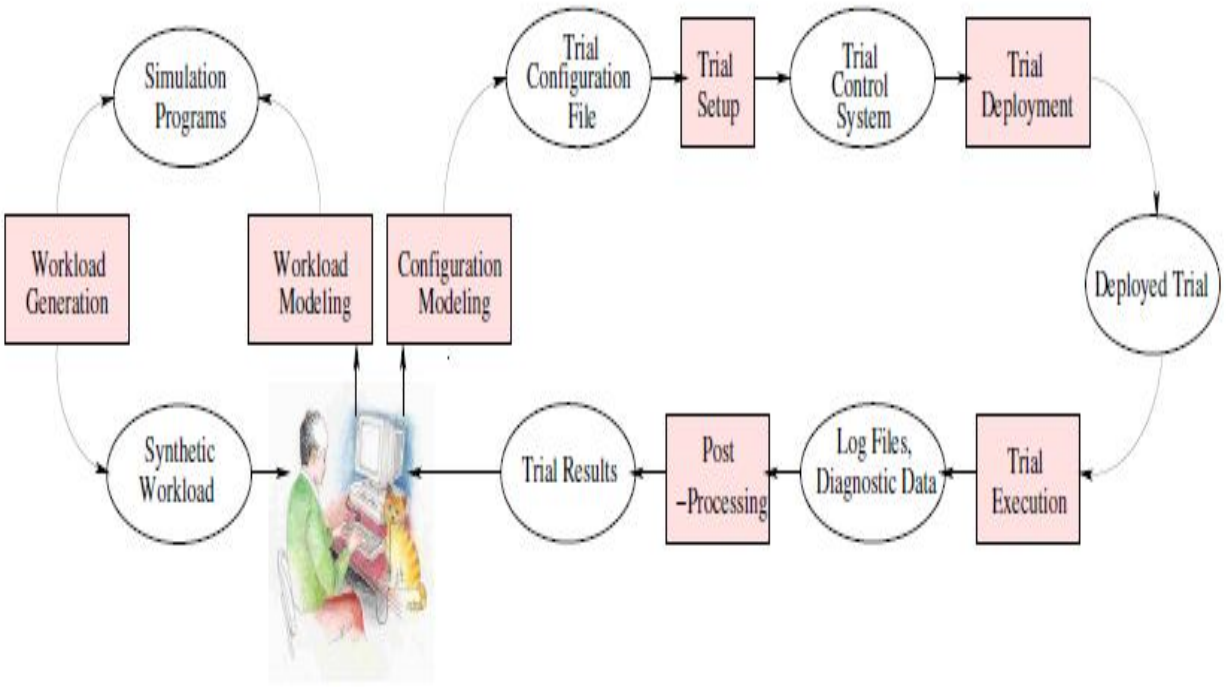

Fig.1.1 Showing the process of automated framework

In the fig. 1.1, As a result, the clients of a distributed system are also highly distributed geographically. It is a big challenge to coordinate execution and communication of these remote resources in the experiments. The distributed property of a distributed system also poses a big challenge in handling distributed components' execution status and experiment results, which are recorded during experiments.

2. Related Work

For the challenges early on in the development process, formal methods and simulations can offer valuable guidance to the engineers. The formal method verifies distributed designs using arguments based on mathematical models. Example technologies of the formal method are formal specification, model checking, and performance modeling. Another more flexible and thus more popular method, simulation, is based on operational models of a distributed system design, the system's clients, and operating environments. Although both formal methods and simulations can provide valuable information about high-level design alternatives, the abstract models that they are based on often fail to capture important details of actual system behavior. Thus, they are not the proper solutions to the difficulties typical of the later development stages, in which details and accuracy become essential. Instead, software engineers must conduct systematic, repeated experimentation with executable prototypes

of a distributed system in realistic execution environments to yield more accurate results. An experiment in this context is "a rigorous, controlled investigation of an activity, where key factors are identified and manipulated to document their effects on the outcome. Specifically, each experiment consists of a number of closely related trials of a particular distributed system with different values for the key factors. For instance, an experiment aimed at evaluating the performance of a web proxy might consist of a number of trials whose key factors are the parameters controlling its caching policy. Hence of we have gone for the associated with separation of concerns is the concept of abstraction. To abstract means to execution phase can be reflected in the experiment results. Since the execution of a distributed system is a non-linear process, even if a failed operation is re-conducted successfully, the system state modification caused by the previous failed operation may still exist. Designers should concentrate on the elaboration of designs, using the specification language merely as a vehicle for the representation of design characteristics ignore characteristics of an object which are not relevant from some point of view? Different hierarchically related points of view determine different abstraction levels. In design methodologies, the object we consider is the system to be designed. At the beginning of the design trajectory we abstract from design concerns that determine the irrelevant details of the construction of the system. These design concerns become relevant throughout the design trajectory. This allows one to structure the design trajectory according to well-defined goals and activities, i.e. providing these construction details step-by-step, until the realization of the system are obtained. Errors can arise any time in a trial, and their effects on the experiment results are different. The trial execution phase is meant to study the performance of the operation. The system states along the whole A specification language can only be a useful general purpose language if its language elements, syntax and semantics, are defined based on the needs of those who are supposed to use this specification language in the elaboration of designs. The formal semantics of a specification language has the primary goal of allowing one to compare specifications, such that these specifications can be distinguished or considered identical.

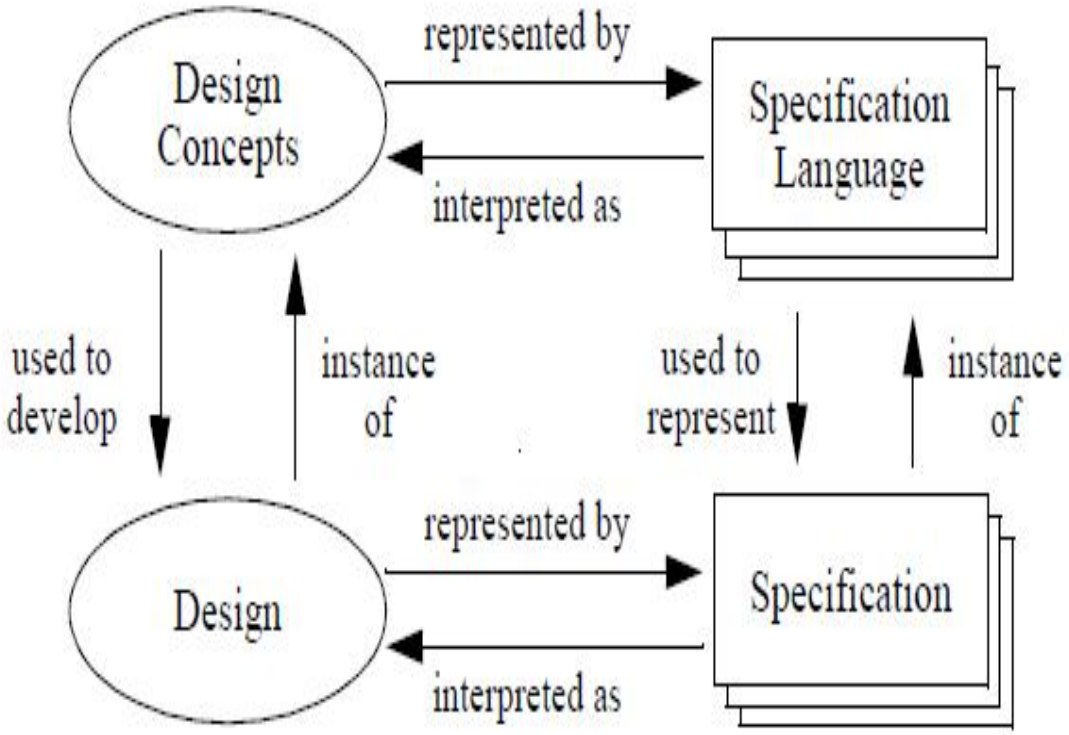

Fig. 2.1 Process of relation between Design and Specification.

In the fig.2.1, it depicts the distinction between a design and its specification, and their relationship with design concepts and specification language. The increasing number of abstraction levels in the representation of designs can be compared with the introduction of programming languages and compilers to replace hand coding in machine code in the early years of computer science. In both cases the objective has been to define concepts which are closer to the intellectual capabilities of human beings than their implementation forms. Once these concepts are introduced, they can be used as building blocks for the definition of more complex concepts. This can be vital for a company since it may cause the loss of competition with respect to other companies that may not structure the design process so well, but are able to reach the concrete products in a shorter time scale. Defining too few design steps may make these steps too complex and are therefore bound to contain errors.

3. Methods

In the environment of industrial competition, the introduction of high level design concepts is often considered ineffective. Normally only the burden of the many design concerns and many alternative orders in which these can be handled is felt, while the benefits brought on by a systematic approach and the potential

improvement on the quality of the final product are not recognized. Very often these benefits are not exploited due to lack of insight. Many real issues require the ability to conduct this experimentation process for distributed systems as quickly as possible. The growth of software development tools has helped software engineers dramatically improve their productivity in software development. They can now deliver software faster. But this also increases the pressure on the software evaluation activity. They need to dramatically improve their efficiency in conducting experiments. Furthermore, rapid and unpredictable change of network status will threaten to make the experimentation results obsolete before the experiment is conducted. For example, some assumptions about traffic mix, topology or protocols of the network might only be valid for less than a few months.

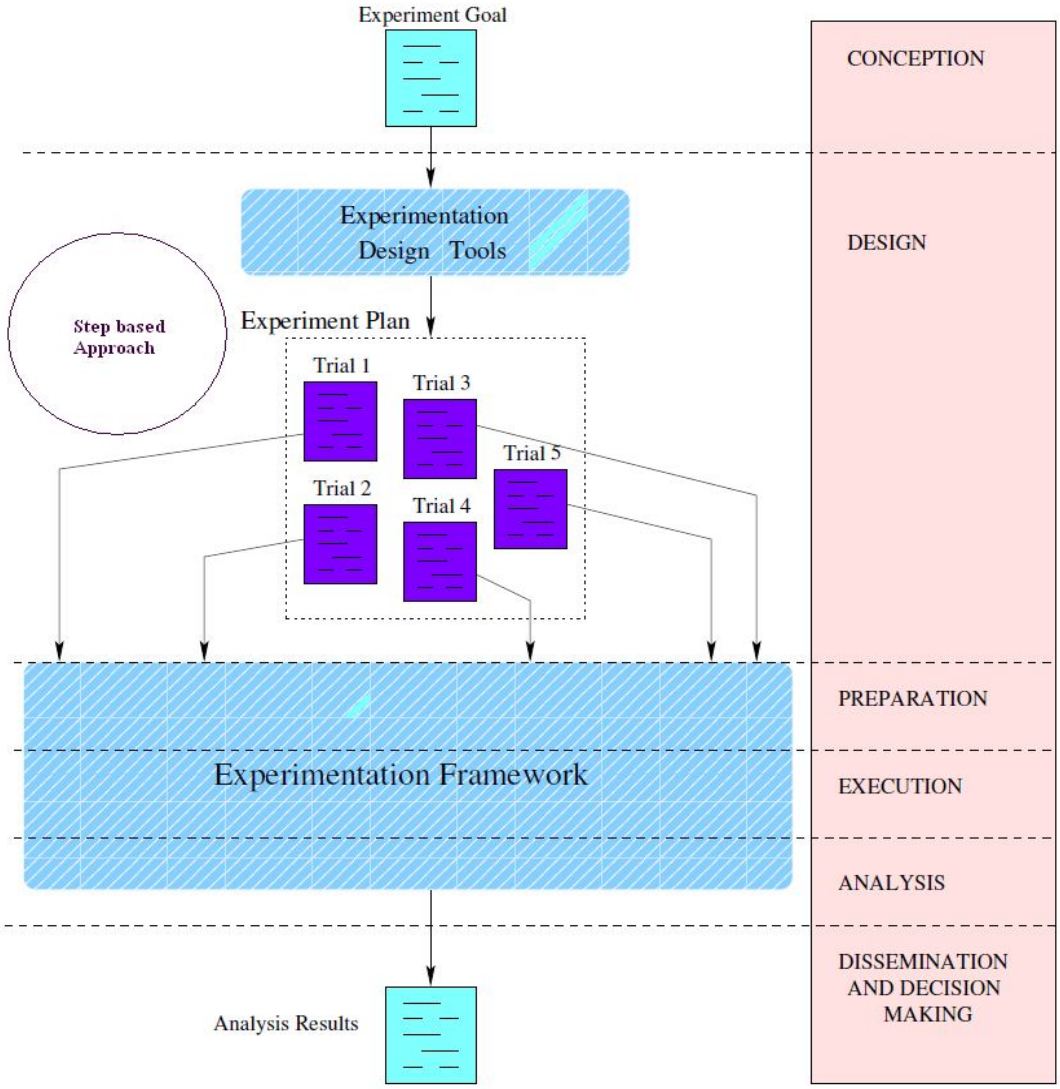

Fig.3.1 Architecture step wise process to experiment the software process of the framework

In the fig.3.1, describes the process of preparation, analysis, execution etc.In this paper, we learn from the related work and create a consistent, unified framework to automate the experimentation activity in engineering distributed systems. We have two hypotheses in building up this framework. First, we proposed a simulation-based workload generation approach to model workload scenarios using discrete-event simulation. The workload can then be generated by running the simulation program, with the execution trace serving as the synthetic workload. Second, we predicted that any experiments with distributed systems can be abstracted in a suite of models. By configuring these higher-level models, an experiment trial can be specified. The low-level, customized trial scripts automating the whole trial process can be automatically generated using generative techniques. We call this approach model based generative approach. Based on these two hypotheses, our experimentation framework can facilitate experimentation activity for distributed systems by providing engineers with a flexible, configurable, and automated, thus, repeatable process for evaluating their distributed systems on distributed test beds.

Generative Programming is "a software engineering paradigm based on modeling software system families such that, given a particular requirement specification, a highly customized and optimized intermediate or end-product can be automatically manufactured on demand from elementary, reusable implementation components by means of configuration knowledge. Instead of building from scratch, it generates specific software systems based on a common generative domain model for the family of these systems. The generative programming paradigm provides us a new way of thinking. There exists a great amount of repetition in experiment script programming for different distributed system experiments. These scripts operate as members of a family sharing the common generative domain model for distributed system experiments. If we have the generative domain model available, the process to produce concrete scripts for a specific experiment is automated and straightforward through a generator.

Evaluation

In this paper, we have thoroughly evaluated our implemented prototype, using case studies and meaningful experiments. Specifically, our evaluation is divided into the following parts.

Fidelity of the workload models: Simulation based workload generation approach can really support software engineers in modeling different distributed system experimental scenarios.

Versatility of the trial configuration models: In this, a suite of trial configuration models is able to define and fully describe an experiment trial with a distributed system.

Scalability of the deployment and execution mechanisms: The realistic experimental analysis of scalability in Chord than that of a published study by performing an experiment that used a similar number of distributed components deployed over an order of magnitude more machines.

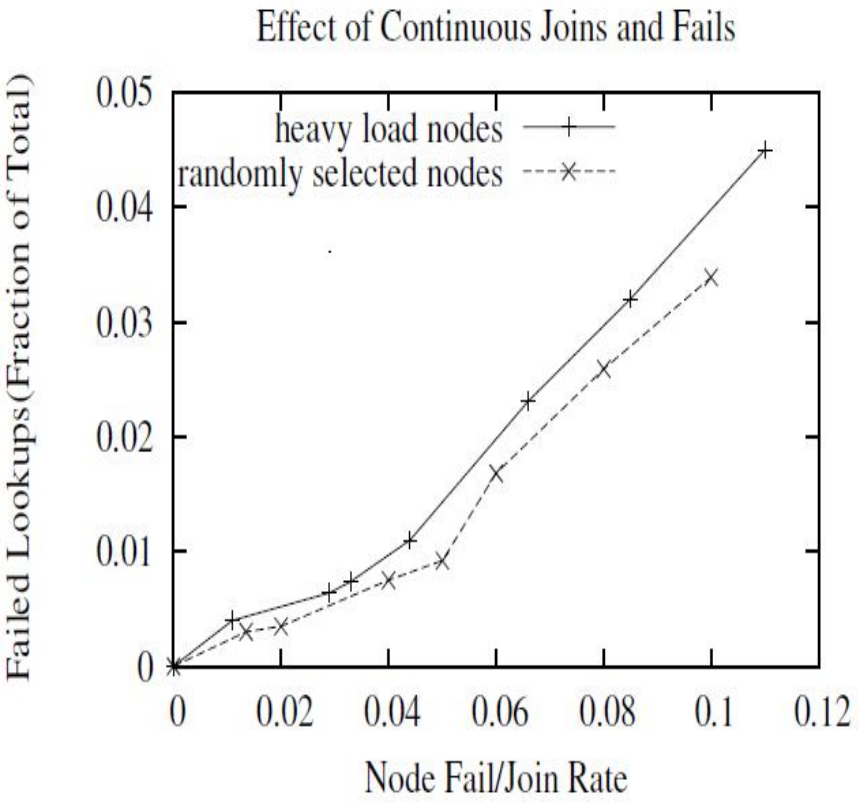

Fig. 3.2 Median factor as function of rate at which nodes are joining at the routing point.

In the fig 3.2, it has been observed higher retrieve failure rates in this experiment than the corresponding results from the last experiment, in which we randomly chose nodes to fail and join. It validates our prediction that the failure of heavier loaded nodes has more effect on the performance of the network.

4. Conclusion

In this paper, we have presented a framework for automating experimentation with distributed systems on distributed test beds. It is targeted at different types of highly distributed systems, removing many practical obstacles, such as scale and heterogeneity that hinder their experimentation in real environments. We consider an experiment made up of a set of closely related trials. The automation framework works at the trial level. It automates the three key steps of each experiment trial: workload generation, trial deployment and execution, and trial post-processing. We designed our approaches for the first two steps, namely the simulation-based workload generation approach and the model-based generative approach. The simulation based approach offers a flexible, complementary workload generation means to the widely used analytical approaches. The model-based generative approach provides a higher-level automation service than other available experimental tools by automating the trial control system construction based on the configuration

modeling of each experiment trial. In summary, through this paper work, we made the following contributions to the research in promoting the experimentation activity of distributed systems: We proposed an approach to categorize distributed actor behaviors based on their dependency on other actor behaviors or on the real system execution.

5. Reference

[1] R. M. Smith and K. S. Trivedi. The analysis of computer systems using markov reward processes. Stochastic Analysis of Computer and Communication Systems, H. Takagi (ed.), pages 589–629, 1990. Elsevier Science Publishers B.V. (North-Holland).

[2] K. S. Trivedi. Probability & Statistics with Reliability, Queuing and Computer Science Applications. John Wiley & Sons, New York, second edition, 2001.

[3] O. C. Ibe, H. C. Choi, and K. S. Trivedi. Performance evaluation of client-server systems. IEEE Transactions on Parallel and Distributed Systems, 4(11):1217–1229, November 1993.

[4] J. F. Meyer. On evaluating the performability of degradable computer systems. IEEE Transactions on Computers, 29(8):720–731, 1980.

[5] L. E. Moser, P. M. Melliar-Smith, and P. Narasimhan. A fault tolerance framework for CORBA. In Proc. of International Symposium on Fault-Tolerant Computing (FTCS-29), pages 150–157, Madison, Wisconsin, June 15-18 1999.

[6] A. Puliafito, S. Riccobene, and M. Scarpa. Modelling of client-server systems. In Proc. of the Third International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS ’95), pages 340–344, Durham, NC, USA, January 18-20 1995.

[7] S. Ramani, B. Dasarathy, and K. S. Trivedi. Reliable messaging using the CORBA Notification service. In Proc. of 3rd International Symposium on Distributed Objects and Applications (DOA ’01), pages 229–238, Rome, Italy, September 18-20 2001.

[8] S. Ramani, K. S. Trivedi, and B. Dasarathy. Performance analysis of the CORBA Notification service. In Proc. of 20thIEEE Symposium on Reliable Distributed Systems (SRDS ’01), pages 227–236, New Orleans, USA, October 28-31 2001.