A Survey of Performance Evaluation Models for Distributed Software System Architecture

Olabiyisi S.O, Omidiora E.O, Uzoka F.M.E, Victor Mbarika and Akinnuwesi B.A

***Abstract* - Over the years, several models were proposed to analyze the performance of distributed software system architecture (DSSA) with the view of avoiding the pitfalls of poor Quality of Service (QoS). In this paper we present a review of research done in this domain for a decade (1999 – 2009) with the view of discovering similarities, differences, merits, implementation and limitations of the performance evaluation models. Based on the analysis, we discovered that these models are machine and software process oriented. The parameters for evaluation are drawn from the software components and processes vis-à-vis the computer machine parameters, for example, the number of processes the CPU has to execute within the limited processor speed. None of the models draws parameters from the contributions of the client organization and end users. Thus this study proposes research direction that focuses on development of models that will have both objective and subjective variables of the client and end users as input parameters to evaluate the performance of software architecture.**

Keywords: Distributed software system, Performance, Performance evaluation model, Software system architecture

1. Introduction

Distributed software systems (DSS) are today one of the complex artifact simultaneously used by many people in the real time operations such as electronic commerce, electronic banking, online payment, and lots of others [49]. Distributed computing is becoming increasingly used as enabling technology for modern enterprise applications, therefore in the face of globalization and ever increasing competition, Quality of Service (QoS) requirements like performance, security, reliability, and robustness are of crucial importance [50]. Organizations must ensure that the DSS they operate does not only provide all relevant functional services, but also meet the performance expectation of their customers. Thus it is important to analyze and predict the expected performance of distributed software systems in order to avoid the pitfalls of poor QoS.

S.O. Olabiyisi and E.O Omidiora are with Department of Computer Science and Engineering, LAUTECH, Ogbomosho, Nigeria.(e-mail:tundeolabiyisi@hotmail.com, omidiorasayo@yahoo.co.uk)

F.M.E. Uzoka is with Department of Computer Science, Mount Royal University, Calgary, Canada (e-mail: uzokafm@yahoo.com)

Victor Mbarika is a Professor of Management Information System and the Director of ICITD in Southern University, Baton Rouge, Louisiana, U.S.A. (e-mail: victor@mbarika.com)

B.A. Akinnuwesi is with Department of Information Technology, Bells University of Technology, Ota, Ogun State, Nigeria and a PhD student with Department of Computer Science and Engineering, LAUTECH, Ogbomoso, Nigeria. (e-mail: akinboluade@yahoo.com)

This research is supported by Bells University of Technology, Ota, Ogun State, Nigeria and the International Centre for Information Technology and Development (ICITD), Southern University, Baton Rouge, Louisiana, U.S.A.

Software architecture (SA) describes how a system is decomposed into components, how these components are interconnected and how they communicate and interact with each other. These aspects of software design are major sources of errors; therefore they have to be well understood. The architecture of a software system is composed of the following [2]:

i. **Conceptual architecture:** describes the system in terms of its major design elements and the relationships among them.
ii. **Module interconnection architecture:** encompasses two orthogonal structures – functional decomposition and layers.
iii. **Execution architecture:** describes the dynamic structure of the system.
iv. **Code architecture:** describes how the source code, binaries, and libraries are organized in the development environment.

There are two parts to SA; the macro-architecture which focuses on the environment of the software system, and the micro-architecture which covers the internal structure of the software system [1]. SA is an important phase in software life cycle as it allows taking early decisions about a system. Moreover it is the earliest point and highest level of abstraction whereupon useful analysis of software system is possible [18]. Hence performance analysis at this level can be useful for assessing whether a proposed architecture satisfies the end users' requirements, meets the desired performance specifications and helps in making architectural decisions. It also helps to identify potential risks and verify that the quality requirements have been addressed in the design and thus saving major potential modifications later in the software development life cycle or tuning the system after deployment. The architecture of software system is analyzed with the aim of establishing the principal effects of the architecture and thus predicts the quality of the system before it is built. SA is considered the first product in an architecture-based development process and from this point of view, the evaluation at this level should reveal requirement conflicts and incomplete design descriptions from a particular stakeholder's perspective [1].

Performance is a quality attribute that describe the system throughput, responsiveness, resource utilization, turnaround time, latency, failure rate and fault tolerance. Thus assessing and optimizing system performance is essential for the smooth and efficient operation of the software system. There are several approaches for evaluating the performance of system architecture. One of the earliest approaches is the fix-it-later approach [51] which advocates software correctness, and deferring performance considerations to the integration testing phase. If performance problems are detected then, additional hardware may be procured or the software will

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

be tuned to correct the problems. This approach has some limitations such as: it takes time to procure and install hardware and to tune software. Tuning may distort the original software design, testing must be repeated after code changes and interim period of poor performance leaves a negative impression with users long after it is corrected. Though the rational for fix-it-later approach is to save development time and cost but this will not be realized, however, if initial performance is unsatisfactory because of additional time and cost of tuning and maintenance. Also [51] in cooperation with J.C. Browne at University of Texas, proposed ADEPT (A Design-Based Evaluation and Prediction Technique), an analysis technique used in conjunction with the performance engineering discipline. ADEPT was the strategy used to combat the fix-it-later principle and supported the performance engineering process. ADEPT evaluate performance of information system early in the life cycle using specifications for both expected resources requirement and upper bounds. If the performance goal is satisfied for the upper bound, the system design is likely to be stable. ADEPT had the following challenges: lack of automatic feedback component, not robust to evaluate large and complex systems (additional sophisticated analysis tools are required in the case), inability to eliminate unwanted argument in the course of evaluation and inability to work in concurrent processing environment.

In the recent times several models have been developed to constantly evaluate the performance of DSSA; thus this survey puts the developments in a decade (1999 – 2009) in the same perspective by reviewing the state of the models and therefore discovering the following: similarities, differences, merits, implementation, parameters for evaluation and limitations. The study proposes research directions based on this analysis.

2. Software Life Cycle Processes

When building a product or system, it is important to go through a series of predictable steps — a road map that helps to create a timely, high-quality result. The road map is called a software process. Technically a software process is a framework or model for the tasks that are required to build high-quality software. It defines the approach that is taken as software is engineered [3]. The software process model is chosen based on the nature of the project and application, the methods and tools to be used and the controls and deliverables that are required.

All software development can be characterized as a problem solving loop (Figure 1) in which four distinct stages are encountered: status quo, problem definition, technical development, and solution integration [5]. Status quo represents the current state of affairs; problem definition identifies the specific problem to be solved; technical development solves the problem through the application of some technology, and solution integration delivers the results (for example, documents, programs, data, new business function, new product) to those who requested the solution in the first place. This problem solving loop applies to software engineering work at many different levels of resolution. It can be used at the macro level when the entire application is considered, at a mid-level when program components are being engineered, and even at the line of code level. Therefore, a fractal representation can be used to provide an idealized view of process.

Figure 2 presents system life cycle processes as defined in ISO/IEC 12207. The processes are grouped into three broad classes: primary; supporting; and organizational [7]. Primary processes are the prime movers in the life cycle. Supporting processes support another process in performing a specialized function. An organization employs an organizational process to establish, control, and improve a life cycle process. Each process is further designed in terms of its own constituent activities, each of which is further designed in terms of its constituent tasks. An activity under a process is a set of cohesive tasks.

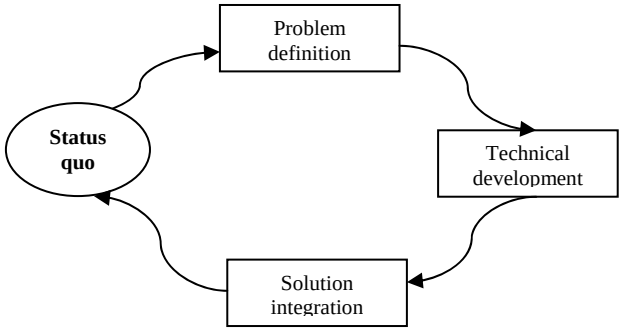

Figure 1 The phases of a problem solving loop [5]

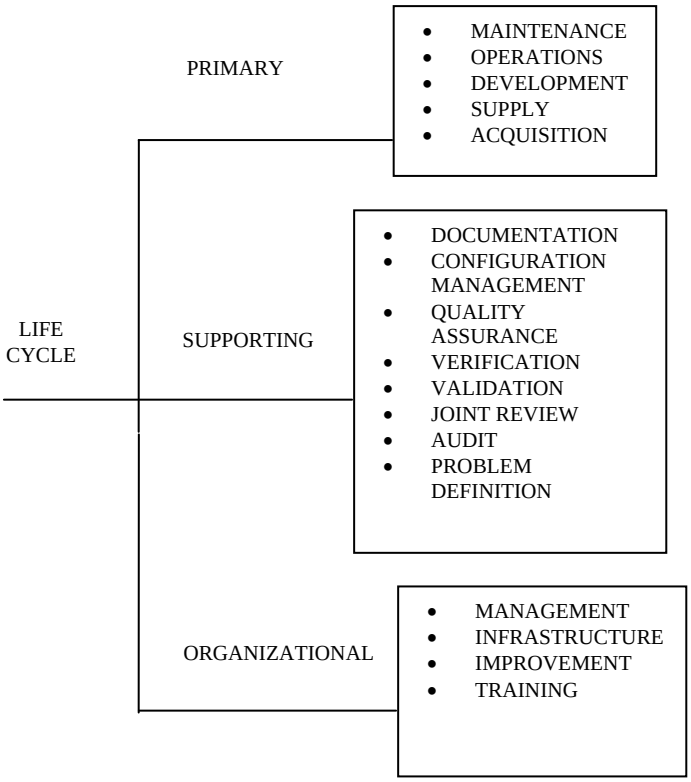

Figure 2 System Life Cycle Processes [7]

3. Related Works

A number of works have been carried out on the survey of system performance evaluation models with the ultimate goal of providing recommendations for future research activities that could significantly improve the performance evaluation and prediction of software system architecture. [8] did a survey of the approaches to

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

software performance from 1960's to 1986. It pointed out the breakthroughs leading to the software performance engineering approach (SPE) and a comprehensive methodology for constructing software to meet performance goals. The concepts, methods, models, tools and use of SPE were summarized and future trend in each areas were suggested. [17] highlighted three indications that concerns software design specifications, performance models and analysis process. The following recommendations were made in the paper: the use of standard software artifacts like UML diagrams for software design specifications; the existence of strong semantic mapping between software artifacts and the performance models as strategy to reduce the performance model complexity and still maintaining a meaningful semantic correspondence; use of simulation besides analytical ones to also address performance model complexity; provision of feedback which is a key success factor for a widespread use of performance analysis models. [52] reviewed performance prediction techniques for component-based software systems and thus made the following recommendations: integration of quantitative prediction techniques in software development process; design of component models allowing quality prediction and building of component technologies supporting quality prediction; inclusion of quality attributes such as reliability, safety or security in the software development process; study of interdependencies among the different quality attributes to determine, for example, how the introduction of performance predictability can affect other attributes such as reliability or maintainability. [1] reviewed eight of the most representative architecture analysis methods at that moment with the view of discovering similarities and differences between these methods by making classifications, comparisons and appropriateness studies. The eight methods considered are: SAAM (Scenario-Based Architecture Analysis Method), SAAMCS (SAAM Founded on Complex Scenarios), ESAAMI (Extended SAAM by Integration in the Domain), SAAMER (Software Architecture Analysis Method for Evolution and Reusability), ATAM (Architecture Trade-Off Analysis Method), SBAR (Scenario-Based Architecture Reengineering), ALPSM (Architecture Level Prediction of Software Maintenance), SAEM (Software Architecture Evaluation Model). The authors discovered at that moment that SAAM has been used for different quality attributes like modifiability, performance, availability and security and it has also been applied in several domains unlike other methods. The other methods were still young and were undergoing refinement and improvement at that moment, thus future work was proposed to evaluate the effects of their various usages and create a repeatable method based on repositories of scenarios, screening and elicitation questions. [53] presented an overview of research in performance modeling, focusing on efforts underway in the Performance Evaluation Research Centre (PERC) and using some new techniques, the authors were able to construct performance models that can be used to project the sustained performance of large-scale scientific programs on different systems, over a range of job and system sizes. Also the model can be used by vendors in system designs, by computing centres in system acquisitions and by application scientists to improve the performance of their codes. [54] defined formal software analyses as having several important properties that distinguish them from other forms of software analysis. Three foundational formal software analyses were described and thus focus on the adaptation of model checking to reason about software. In view of this the authors reviewed emerging trends in software model checking and identifies future directions that promise to significantly improve its cost-effectiveness. [9] did a review on the future of software performance engineering with the view of describing the current progress and future trends within two distinct approaches for predicting and improving software performance: an early-cycle predictive model-based approach and a late-cycle measurement-based approach. The authors recommended convergence of approaches in order to cover the entire development cycle.

4. Classification of Models

This paper surveys and classifies the performance models using the following categories: Factor Analysis, Queuing Network, Petri net, Pattern-Based, UML Based, Hierarchical Modeling, PACE (Performance Analysis and Characterization Environment) Based, Component-Based Modelling, Scenario-Based, Relational Approach, Software Architecture Analysis Methods (SAAM), Aspectual Software Architecture Analysis Methods (ASAAM), Hybrid Approaches such as UML-Petri net, UML-Stochastic Petri net, Queue Petri Nets Approach and Soft Computing Approach.

4.1 Factor Analysis (FA) Based Approach

Factor analysis is a collection of methods used to examine how underlying constructs influence the responses on a number of measured variables. There are basically two types of FA: exploratory and confirmatory. Exploratory factor analysis (EFA) attempts to discover the nature of the constructs influencing a set of responses. Confirmatory factor analysis (CFA) tests whether a specified set of constructs is influencing responses in a predicted way. Both types of FA are based on the Common Factor Model. FA is used mostly for data reduction purposes: to get a small set of variables (preferably uncorrelated) from a large set of variables (most of which are correlated to each other); to create indexes with variables that measure similar things (conceptually) [10]. [55] used FA approach to formulated a model for analysing IT software projects with the view of establishing the success or failure of the project before it takes off. FA as contained in SPSS and Statview software was used. Fifty performance indices of IT projects planning, execution, management and control were formulated. Eleven factors were extracted and subjected to further analysis with a view to estimating and ranking their contribution to the success of IT projects. The model was tested using sample life data gotten from questionnaires that were administered to the principal actors of the popular IT software projects in Nigeria. The significant contribution of the research is the provision of a working model that utilized both quantitative and qualitative decision variables in assessing the success or failure of IT projects. This serves as template for evaluating IT projects prior to its take off.

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

4.2 Queuing Networks

This is a conventional modelling paradigm which consists of a set of interconnected queues [11]. Each queue represents a service station, which serves requests sent by customers. A service station consists of one or more servers and a waiting area which holds requests waiting to be served. When a request arrives at a service station, its service begins immediately if a free server is available. Otherwise, the request is forced to wait in the waiting area (buffer) or the service of another request is pre-empted in case the arriving request has a higher priority. The time between successive request arrivals is called ineterarrival time. Each request demands a certain amount of service, which is specified by the length of time a server is occupied serving it, that is, the service time. The queuing delay is the amount of time the request waits in the waiting area before its service begins. The response time is the total amount of time the request spends at the service station, that is, the sum of the queuing delay and the service time. The models based on Queuing Networks are categorized in Table 1.

Table 1 Queuing Network Based Performance Models

Description of Model	Parameters Considered
[16] designed and implemented object-oriented queuing network model – a reusable performance models for software artifacts.	Buffer size, processor speed of server, queue size, number of incoming request, request arrival time, request departure time.
[17] integrated performance and specification model to provide a tool for quantitative evaluation of software architecture at the design phase.	Number of service centres, service rate of service centre, arrival rate of requests at service centre, number of servers in service centres, routing procedure of requests, Number of request circulating in the system, physical resources available system workloads, network topology
[18] modelled layered software system as a closed Product Form Queuing Network (PFQN) and solve it for finding performance attributes of the system	Range of number of clients accessing the system, average think time of each client, number of layers in the software system, relationship between the machines and software components, number of CPUs and disks on each of the machine and thread limitation (if any), uplink and downlink capacities of the connectors connecting machines running adjacent layers of the system, size of packets of the links, service time required to service one request by a software layer, forward transition probability, rating factors of the CPU and the disks of each machines in the system
[19] proposed an approach based on queuing networks models for performance prediction of software systems at the software architecture level, specified by UML.	Same as above [18]
[20] developed Software Architecture and Model Extraction (SAME) technique that extract communication patterns from executable designs or prototype that use message passing, to develop a Layered Queuing Network Performance Model in an automated fashion.	Same as [18]

4.3 Petri Net Approach

Petri nets were introduced in 1962 by Dr. Carl Adam Petri [24]. A Petri net is a graphical and mathematical modelling tool [12]. It is a directed bipartite graph with an initial state called the *initial marking*. Petri Nets consist of four basic elements: places, transitions, tokens and arcs. Places represent a condition in the process. Transitions represent the stochastic or time-based nature of changes in the model. Transitions can be immediate, deterministically time-delayed, or time-delayed based on a probability distribution defined by the user. Tokens represent objects in the model. When too many tokens appear in one place, most applications revert to placing a number inside the place. Transitions are active components. They model activities which can occur, thus changing the state of the system. Transitions are only allowed to fire if they are *enabled*, which means that all the preconditions for the activity must be fulfilled (that is there are enough tokens available in the input places). When the transition fires, it removes tokens from its input places and adds some at all of its output places. The number of tokens removed / added depends on the cardinality of each arc. The interactive firing of transitions in subsequent markings is called token game. Lastly, arcs determine the path that tokens take throughout the model. Arcs can either enable or inhibit movement in the model, depending on their use. System performance models based on Petri net approach are categorized in Table 2.

Table 2 Petri Net Based Performance Models

Description of model	Parameters Considered
[21] developed performance evaluation model for Agent-based system using petri net approach	System load, system delays, system routing rate, latency of process, CPU time.
[22] Performance analysis of Internet based software retrieval systems using petri nets	Network time.
[23] developed stochastic petri nets model from UML activity diagrams	Routing rate, action duration, system response time.
[25] translated UML activity diagram into stochastic Petri net model that allows to compute performance indices.	Routing rate, action duration, system response time.
[26] derived performance parameters from Generalized Stochastic Petri Net (GSPN) using Markov chain theory.	Routing rate, action duration, system response time.

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

4.4 Queuing Petri Net (QPN)

The hybrid of Petri Net and Queuing Networks is Queuing Petri Nets (QPNs) which facilitates the integration of hardware and software aspects of system behavior into the same model. In addition to hardware contention and scheduling strategies, using QPNs one can easily model simultaneous resource possession, synchronization, blocking and contention for software resources. Thus QPNs combines Queuing Networks and Petri Nets into a single formalism with the view of eliminating their disadvantages. QPNs allow queues to be integrated into places of Petri Nets and this enables the modeler to easily represent scheduling strategies and bring the benefits of Queuing Networks into the world of Petri Nets [11]. System performance models based on Queuing Petri net approach are categorized in Table 3.

Table 3 Queuing Petri Net Based Performance Models

Description of Model	Parameters Considered
[11] apply QPN formalism to analyse the performance of distributed e-business system.	Service demand of queue, service rate of queue, token population of queue, queue size, buffer size, processor speed of server, routing rate.
[27] presented a novel case study of a realistic state-of-the-art distributed component-based system, showing how the QPN modelling formalism can be exploited as software system performance prediction tool.	Same as [11]

4.5 PACE (Performance Analysis and Characterization Environment) Based Approach

The motivation to develop PACE in [13] was to provide quantitative data concerning the performance of sophisticated applications running on high performance systems. The framework of PACE is a methodology based on a layered approach that separates out the software and hardware system components through the use of a parallelization template. This is a modular approach that leads to readily reusable models, which can be interchanged for experimental analysis. Each of the modules in PACE can be described at multiple levels of detail and thus providing a range of result accuracies but at varying costs in terms of prediction evaluation time. PACE is aimed to be used for pre-implementation analysis, such as design or code porting activities as well as for on-the-fly use in scheduling systems. The core component of PACE is a performance specification language, CHIP^{3}S (Characterization Instrumentation for Performance Prediction of Parallel Systems). CHIP^{3}S provides a syntax that allows the description of the performance aspects of an application and its parallelization, to be expressed. This includes control flow information, resource usage information (for example number of operations), communication structures and mapping information for a parallel or distributed system. The software object in the PACE system were created using the Application Characterization Tool (ACT). ACT aids the conversion of sequential or parallel source code into the CHIP^{3}S language via the Stanford Intermediate Format (SUIF). ACT performs a static analysis of the code to produce the control flow of the application, operation count in terms of high-level language operations and also the communication structure. The hardware objects of the model are created using a Hardware Model Configuration Language (HMCL) by specifying system-dependent parameters. On evaluation, the relevant sets of parameters are used and supplied to the evaluation methods for each of the component models.

4.6 Hierarchical Performance Modeling Approach

In [14] a hierarchical performance modeling (HPM) technique for distributed systems which incorporated different level of modeling abstraction was presented. HPM is a technique to model performance for different layers of abstraction. It includes several layers of organization from primitive operation to software architecture, providing a degree of accuracy that cannot be achieved with single layer models. The application is developed in a top-down fashion from general to more specific, but performance information is generated in bottom-up method, thus linking the different levels of analytic models into a composite model. This approach support specification and performance model generation that incorporates computation and communication delays along with hardware profile characteristics to assist in the evaluation of performance alternatives. HPM models provide a quantitative performance assessment of an entire system comprising of hardware, software and communication. The HPM provided a well defined methodology to allow system designers to evaluate the application based on the system requirements of his/her application and fine tune the values of performance parameters.

4.7 Pattern Based Approach

Design patterns are defined as description of communicating objects and classes that are customized to solve a general design problem in a particular context. The components of design pattern are: *Pattern name, Intent, Motivation, Applicability, Structure, Participants, Collaborations, Consequences, Implementation, Sample code, Known uses, Related pattern.*

Performance models based on pattern based approach are presented in Table 4.

4.8 Soft Computing Approach

Soft computing is an approach to computing which parallels the remarkable ability of the human mind to reason and learn in an environment of uncertainty and imprecision [15]. It is a consortium of methodologies centering in fuzzy logic (FL), artificial neural networks (ANN) and evolutionary computation (EC). These methodologies are complementary and synergistic, rather than competitive. They provide in one form or another flexible information processing capability for handling real life ambiguous situations. Soft computing aims to exploit the tolerance for imprecision, uncertainty, approximate reasoning and partial truth in order to achieve tractability, robustness and low-cost solutions. The attributes of these models are often measured in terms linguistic values, such as *very low, low, high and very high*. The imprecise nature of the attributes

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

constitutes uncertainty and vagueness in their (subsequent) interpretation. Thus software performance evaluation models should be able to deal with imprecision and uncertainty associated with such (the) linguistic values. Performance models based on soft computing approach are presented in Table 5. The advantage of softcomputing models particularly fuzzy logic and ANN are [32]: they are more general, they mimic the way in which humans interpret linguistic values and the transition from one linguistic value to a contiguous linguistic value is gradual rather than abrupt.

Table 4 Pattern Based Performance Models

Description of Model	Parameters Considered
[28] presented an approach based on patterns to develop performance models for object oriented software system in the early stages of the software development process. This complement the approach given in [21]	Event load, time to perform an action, request arrival time, request service time, number of concurrent users
[29] presented a pattern-based approach to model the performance of software system and used it to evaluate the performance of mobile agent system	Same as [28]
[31] presented a pattern-based performance completion for message-oriented middleware	System configuration (hardware & network components), message size (incoming & outgoing), delivery time for message, number of message sent, size of message sent, number of message delivered, size of message delivered, transaction/request size, buffer/pool size

Table 5 Performance Models Based Soft Computing Approach

Description of Models	Parameters Considered
[32] applied fuzzy logic to measure similarity of software projects when their attributes are described by categorical values (linguistic values in fuzzy logic)	Seventeen parameters:- software size, project mode plus 15 cost drivers.
[33] presented a new technique based on fuzzy logic, linguistic quantifiers and analogy-based reasoning to estimate the cost of or effort of software projects when they are described by either numerical data or linguistic values.	Same as in [32]
[34] showed how fuzzy logic can be applied to computer performance work to simplify and speed analysis and reporting.	CPU Queue length, memory (RAM) available, pages input per second, read time, write time, I/Os per second.
[35] Developed a fuzzy model for evaluating information system projects based on their present value using fuzzy modelling technique.	Three parameters representing three possible values of project costs, benefits, evaluation periods and discount rate.

4.9 Other Performance Models

In [36] Multivariate Adaptive Regression Splines (MARS) was used for software performance analysis. The parameters considered are the. A resource function was designed and automated, having the following parameters - size of data objects, number of disk blocks to be read, size of messages to be processed, memory and cache size, processor speed, bus and network bandwidth.

In [37] PASA a method for performance assessment of software architectures was developed and it was scenario-based. It uses the principles and techniques of SPE (software performance engineering) to identify potential areas of risk within the architecture with respect to performance and other quality objectives. It identifies strategies for reducing or eliminating the risks if a problem is found. Scenario for important workloads are identified and documented. The scenarios provide means of reasoning about the performance of the software as well as other qualities and they serve as starting point for constructing performance models of the architecture.

ASAAM (Aspectual Software Architecture Analysis Method) was proposed in [38] to identify and specify the concerns at the architecture design level which inherently crosscut multiple architectural components, which cannot be localized in one architectural component and which, as such, cannot be easily managed by using conventional abstraction mechanism. It makes these transparent early in the software development life cycle. ASAAM is scenario-based. It introduces a set of heuristic rules that help to derive architectural aspects and the corresponding tangled architectural components from scenarios. It takes as input the architecture design and measure the impact of predefined scenarios on it in order to identify the potential risks and the sensitive points of the architecture. This helps to predict the quality of the system before it is built, thereby reducing unnecessary maintenance costs.

[39] presented performance analysis based on requirements traceability. Requirement traceability is critical to providing a complete approach which will lead to an executable model for performance evaluation. This paper investigated the software architectures that are extended based on the performance requirements traceability to represent performance property. The extended architecture are then transformed into a simulation model colored GSPN and the simulation results are used to validate performance requirements and evaluate system design. The parameters considered are queue length, number of request to be serviced, server response time, server execution time, processor speed.

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

5. Discussions and Limitations of Existing Models

5.1 Discussion

From this literature review we establish the following:

i. The models are algorithmic using hard computing principles.
ii. Parameters for evaluation are mostly machine centered and they are objective. For example, processor speed, bandwidth size, RAM size, message size.
iii. The models are implemented at the early stage of the software life cycle.
iv. Though the authors acknowledged the contributions of the client company/end users during software development process but none of the models considers the contributions of the management of the client company or the end users as factors that can influence the decisions of the software developer as per system architectural style and pattern.
v. The models are developed using object programming methodology and thus they are re-useable and scalable.
vi. Performance metrics considered are mostly the following: throughput, response time and resource utilization.

5.2 Limitations of the Models

Efficient and effective software architecture is a product of a good collaborative effort of the end users and the software developers which form the software development team. Thus the success of an enterprise software system is a collaborative effort of the users and software developers [40, 41, 42, 43, 44, 45, 46, 47, 48]. If any of the parties is unable to make meaningful and useful contribution, the software performance may not be guaranteed. Moreover, performance is subjective and thus can be interpreted differently by different people. Therefore while evaluating the performance of a software system; the factors contributed by each party have to be put into consideration to ascertain the performance of the system. In these models, the end users' subjective decision variables that could influence the choice of the software architectural style and design pattern made by the software developers are not built in as some of the input parameters for evaluation. Moreover, the models are limited by their inability to cope with uncertainties and imprecision of data or information surrounding software projects early in the development life cycle. In addition, the conceptual structures of some approaches (for example, probabilistic models) that can represent vague information are inadequate for dealing with problems in which information is perception-based and is expressed in a natural language. Also they do not address the problem of semantics of human natural languages describing system performance which in many situations are fuzzy. In addition, the models are computationally intensive [32] and are intolerant of noise and of irrelevant features. They cannot handle categorical data other than binary valued variables. However in measuring software metrics, some users driven factors (that is linguistic variables in fuzzy logic), such as the experience of programmer, involvement of the client/end users in software development process, the complexity of modules; are measured on an ordinal scale composed of qualifications such as very low', 'low', 'nominal', 'high', 'very high', and 'extra-high' (that is linguistic values in fuzzy logic); are important.

6. Conclusion and Future Work

6.1 Conclusion

Software performance is a pervasive quality of software systems and it is affected by the software itself and all underlying layers, such as operating system, middleware, hardware, communication networks, client and end users involvement in software life cycle process. Thus in this paper we have reviewed research on performance evaluation models from 1999 to 2009 with the view of establishing the trend of the models within this period, their underlying principles of design and implementation and their limitations. We were able to establish that the parameters for the performance models in this period were machine centered/driven and we therefore propose that future models should have as input parameters, the linguistic decision variables of users that affect the choice of software architectural style and pattern.

6.2 Future Work

Client organization and the end users are key players in software development process. Therefore decision variables such as commitment of the staff of the client company; IT literacy level of operations staff of the client company; adequacy of user requirement specification and representation; communication between users and software developers; fund availability; technical know-how of operations staff and lots of more should not be underestimated while establishing the variables to evaluate software architecture. In view of this we propose that future works should identify both objective and subjective decision variables peculiar to the users of software systems (that is management staff of the client organization, the operational staff and other end users) that influence the choice of architectural style and design pattern made by the software developer. Developing a performance model that can capture these users' decision variables will help making system performance evaluation to be more users driven and thus complement the existing models that are mostly driven by machine parameters.

Reference

[1] Dobrica L. and Niemela E (2002). A Survey on Software Architecture Analysis Methods. IEEE Transactions on Software Engineering, Vol. 28, No. 7, July 2002
[2] Soni, D., Nord, R., and Hofmeister, C. (1995). Software Architecture in Industial Applications. In Proceedings 17th International Conference on Software Engineering (ICSE17), pages 196--207. IEEE.
[3] Pressman Roger S. (2001). Software Engineering. Fifth edition. Published by McGraw-Hill, a division of The McGraw-Hill Companies, Inc. 1221 Avenue of the Americas, New York, NY, 10020
[4] ISO/IEC 15288, The System Life Cycle Process Standar for the 21st Century. Available @ http://syseng.omg.org/ ISOIEC15288.pdf
[5] Raccoon, L.B.S., "The Chaos Model and the Chaos Life Cycle," *ACM Software Engineering Notes*, vol. 20., no. 1, January, 1995, pp. 55–66.
[6] Raghu Singh (1995). An introduction to ISO/IEC 12207 (Tutorial), August 1995.

[7] Raghu Singh. International Standard ISO/IEC 12207 Software Life Cycle Processes. URL: http://www.abelia.com/docs/12207cpt.pdf
[8] Connie U. S., (1986). The Evolution of Software Performance Engineering: A Survey. In proceedings of 1986 ACM Fall Joint Computer Conference. Pp 778 – 783.
[9] Woodside M., Franks G., Petriu D.C (2007). The Future of Software Performance Engineering. In IEEE Proceeding of Future of Software Performance Engineering (FOSE'07)
[10] DeCoster Jamie (1998). Overview of Factor Analysis. Available @ http://www.stat-help.com/notes.html
[11] Samuel Kounev and Alejandro Buchmann (2003). Performance Modeling of Distributed E-Business Applications Using Queuing Petri Nets. In proceedings of IEEE International Symposium on Performance Analysis of Systems and Software, March. 145 – 153
[12] Peterson, James Lyle (1981). Petri Net Theory and the Modeling of Systems. Prentice Hall, ISBN 0-13-661983-5.
[13] Junwei Cao, Darren J. Kerbyson, Efstathios Papaefstathiou and Graham R. Nudd (2000). Performance Modeling of Parallel and Distributed Computing Using PACE. In proceedings of IEEE International Performance Computing and Communications Conference, IPCCC-2000, Phoenix, February. 485 – 492.
[14] Smarkusky D., Ammar I. Antonios and Sholi H. (2000) Hierarchical Performance Modeling for Distributed System Architecture. Available @ http://www.cs.sfu.ca/~mhefeeda/papers/ISC2000-HPM.pdf
[15] Gary R. George P. E. and Frank Cardullo (1999). Application of Neuro-Fuzzy Systems to Behavioral Representation in Computer Generated Forces. In proceedings of 8th Conference on Computer Generated Forces and Behavioural Representation (Orlando FL, May 11-13, 1999)
[16] Savino-Vazquez Nunzio-Nicolo and Puigjaner Ramon (2001). A Component Model for Object-Oriented Queuing Networks and its Integration in a Design Technique for Performance Models. In Proceedings of the 2001 Symposium on Performance Evaluation of Computer and Telecommunication System (SPECTS 2001) in Orlando, Florida, July 15 – 19.
[17] Simonetta Balsamo, Roberto Mamprin and Moreno Marzolla (2004). Performance Evaluation of Software Architecture with Queuing Networking Model. In proceedings of ESMc'04, Paris France, October 25 – 27.
[18] Vibhu Shanjanya Sharma, Pankaj Jalote, Kishor S. Trivedi, (2005). "Evaluating Performance Attribute of Layered Software Architecture". Paper published in CBSE 2005: Refer LNCS 3489, pp 66-81.
[19] Simonetta Balsamo and Moreno Marzolla, 2005. Performance Evaluation of UML Software Architectures with Multiclass Queueing Network Models. WOSP'05 July 12 – 14, Palma de Mallorca, Spain. Copyright 2005 ACM 1-59593-087-6/05/0007.
[20] Israr A. Tauseef, Lau H. Danny, Franks Greg and Woodside Murray (2005). "Automatic Generation of Layered Queuing Software Performance Models from Commonly Available Traces". WOSP'05 July 12 – 14, Palma de Mallorca, Spain. Copyright 2005 ACM 1-59593-087-6/05/0007
[21] Merseguer Jose, Javier Campose and Eduardo Mena (2000). Performance Evaluation for the Design of Agent-Based Systems: A Petri Net Approach. In proceedings of the workshop on Software Engineering and Petri Nets within the 21st International Conference on Application and Theory of Petri Nets, University of Aarhus. 1 – 20.
[22] Merseguer Jose, Campos Javier and Mena Eduardo, 2001. Performance Analysis of Internet Based Software Retrieval Systems Using Petri Nets. In proceedings of 4th ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile System 2001, Rome Italy
[23] Juan Pablo Lopez-Grao, Jose Merseguer and Javier Campos, 2004. From UML Activity Diagrams to Stochastic Petri Nets: Application to Software Performance Engineering. WOSP'04 January 14-16, 2004, Redwood City, California. Copyright 2004 ACM 1-58113-673-0/04/0001.
[24] Petri C.A. 1962. Kommunikation mit Automaten. English Translation, 1966: Communication with Automata. Technical Report RADC-TR-65-377, Rome Air Dev. Centre, New York.
[25] Juan Pablo Lopez-Grao, Jose Merseguer and Javier Campos, 2008. On the use of Formal Models in Software Performance Evaluation. News in the Petri Nets World, Dec. 27. URL: http://webdiis.univzar.es.crpetri/paper/jcampos/02_LGMC_JJCC.pdf
[26] Motameni H., Movaghar A., Siasifar M., Montazeri H. and Rezaei A. (2008). Analytic Evaluation on Petri Net by Using Markov Chain Theory to Achieve Optimal Models. World Applied Sciences Journal Vol. 3, No. 3, pp 504 – 513.
[27] Samuel Kounev (2006). Performance Modeling and Evaluation of Distributed Component-Based System Using Queuing Petri Nets. IEEE Transactions on Software Engineering. 32, No. 7, 487 – 502.
[28] Merseguer Jose, Javier Campose and Eduardo Mena (2000a). A Pattern-Based Approach to Model Software Performance. In proceedings of the 2nd International Workshop on Software and Performance, Ottawa, Ontario, Canada. 137 – 142.
[29] Merseguer Jose, Javier Campose and Eduardo Mena (2003). A Pattern-based Approach to Model Software Performance Using UML and Petri Nets: Application to Agent-based Systems. Proceedings of 7th World Multiconference on Systemic Cybernetics and Informatics, Orlando, Florida, USA. Vol. 9, pp 307 – 313
[30] Fanjiang Yong-Yi, Hsueh Nien-Lin and Lee Jonathan (2005). A Pattern-based Model Transformation Approach to Enhance Design Quality. Available @ www.atlantis-press.com/php/download-paper.php?id=65
[31] Happe Jens, Friedrich Holger, Becker Steffen and Reussner H. Ralf, 2008. A Pattern-Based Performance Completion for Message-Oriented Middleware. WOSP'08, June 24-26, Princeton, New Jersey, USA. Copyright 2008 ACM 978-1-59593-873-2/08/06.
[32] Idris Ali and Abran Alain (2001). A Fuzzy Based Set of Measures for Software Project Similarity: Validation and Possible Improvements. Published in METRICS 2001, London, England. Pp 85 – 96.
[33] Ali Idris, Alain Abran and Khoshgoftaar, 2004. Fuzzy Case-Based Reasoning Models for Software Cost Estimation. Published in Springer-Verlag 2004. Available @ http://www.gelog.etsmtl.ca/publications/pdf/803.pdf
[34] Maddox Michael, 2005. Using Fuzzy Logic to Automate Performance Analyses. In proceedings of the Computer Measurement Group's 2005 International Conference. Copyright 2005 by The Computer Measurement Group inc.
[35] Omitaomu A. Oluwafemi and Adedeji Badiru, 2007. Fuzzy Present Value Analysis Model for Evaluating Information System Projects. Published in the Engineering Economist, Vol. 52, Issue 2, pp 157 – 178.
[36] Courtois Marc and Woodside Murray, 2000. Using Regression Splines for Software Performance Analysis.

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

WOSP 2000, Ontario, Canada. Copyright ACM 2000 1-58113-X/00/09

[37] Lloyd G. Williams and Connie U. Smith, 2002. PASA[SM]: An Architectural Approach to Fixing Software Performance Problems. Copyright 2002, Software Engineering Research and Performance Engineering Services.

[38] Tekinerdogan Bedir, 2003. ASAAM: Aspectual Software Architecture Analysis Method

[39] Wise Jeffrey C., Chang Carl K., Xia Jinchun and Cleland-Huang Jane, 2005. Performance Analysis Based on Requirements Traceability. Technical Report 05 – 04, Dept of Computer Science, Iowa State University.

[40] Sari Kujala, Marjo Kauppinen and Sanna Rekola, 2001. Bridging the Gap Between User Needs and User Requirements. In proceedings of PC-HCI 2001 Conference Patras, Greece.

[41] Blake Ives, Margrethe H. Olson and Jack J. Baroudi, 1983. The Measurement of User Satisfaction. Communications of the ACM, Vol. 26 No. 10. Pp 785 – 793

[42] Procaccino J. Drew and Verner M. June, 2009. Software Developers' Views of End-Users and Project Success. Communications of the ACM, Vol. 52, No. 5. Pp 113 – 116.

[43] Muthitacharoen Achita and Saeed A. Khawa, 2009. Examining User Involvement in Continuous Software Development (A Case of Error Reporting System). Communications of the ACM, Vol. 52, No. 9. Pp 113 – 117.

[44] Dodd James L and Carr Houston H.I, 1994. Systems Development Led by End-Users: An Assessment of End-User Involvement in Information Systems Development. Journal of Systems Management.

[45] Daniel Robey and Dana Farrow, 1982. User Involvement in Information System Development: A Conflict Model and Empirical Test. Journal of Management Science, Vol. 28, No. 1

[46] Serkan CAK and Kursat Cagiltay, 2005. Bridging the Gap Between Users and Developers in Software Intensive Projects of Turkish Defense Industry. URL http://www.metu.edu.tr/~kursat/Cak&Cagiltay.pdf

[47] Procaccino J. Drew, Verner M. June and Lorenzet J. Steven, 2006. Defining and Contributing to Software Development Success. Communications of the ACM, Vol. 49 No. 8

[48] Brian Withworth, Jerry Fjermestad and Edward Mahinda, 2006. The Web of System Performance. Communications of the ACM, Vol. 49, No. 5. Pp 93 – 99

[49] Merseguer Jose and Javier Campos (2004). Software Performance Modeling Using UML and Petri Nets, LNCS2965, Springer Verlag. 265-289.

[50] Samuel Kounev (2006). Performance Modeling and Evaluation of Distributed Component-Based System Using Queuing Petri Nets. IEEE Transactions on Software Engineering. 32, No. 7, 487 – 502.

[51] Connie U. Smith, 1981. Increasing Information System Productivity. In proceedings of the Computer Measurement Group's 1981 International Conference. Copyright 1981 by The Computer Measurement Group Inc.

[52] Steffen Becker, Lars Grunske, Raffaela Mirandola, and Sven Overhage, 2006. Performance Prediction of Component-Based Systems: A Survey from an Engineering Perspective. In Architecting Systems with Trustworthy Components, volume 3938 of LNCS, pages 169–192. Springer.

[53] Bailey H. David and Snavely Allan, 2005. Performance Modeling: Understanding the Present and Predicting the Future. Proceedings of Euro-Par, Lisbon, Portugal.

[54] Dwyer B. Mathew, Hatcliff John, Pasareanu S. Corina and Visser Willen, 2007. Formal Software Analysis: Emerging Trends in Software Model Checking. Future of Software Engineering (FOSE'07). Copyright IEEE.

[55] Chiemeke S. Chinye (2003). Computer Aided System for Evaluating Information Technology Projects, PhD thesis submitted to the School of Postgraduate Studies, Federal University of Technology, Akure, Ondo State, Nigeria.

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)