Investigation of the paper: A Method for Design and Performance Modeling of Client/Server Systems

Ekrem Kocaguneli, Katerina Goseva
CS & EE, WVU, Morgantown, USA
ekocagun@mix.wvu.edu,
katerina.goseva@mail.wvu.edu

ABSTRACT

Problem: It is hard for performance engineers to design complex and distributed client/server (C/S) applications that can meet the performance goals.
Aim: When design and performance modeling activities are combined, they will help one another and lead to better desing of C/S systems.
Method: A performance engineering language developed by the authors map the use cases to performance specifications and generates an analytical performance model for the system. Service demands at servers, storage boxes and the network are also generated from the system specifications.
Results: As an example the query optimizer of a DBMS was modeled to be able to generate more accurate estimates on the I/Os. They have observed a clear effect of improvement when the database configuration was changed in accordance with the suggestions of the proposed method.
Conclusion: A method that integrated design and performance modeling activities was proposed and was applied on a practical system. The model enabled early predictive performance models and the parameters.

Categories and Subject Descriptors

Software Engineering [**Software Performance Engineering**]: Client Server Systems

General Terms

Software Performance Engineering

Keywords

Performance Models, Client Server Systems

1. INTRODUCTION

The paper will be organized in the following manner: I will follow the original paper [4] and their experiments while discussing the topic and from time to time I will branch out to other papers in the literature that have a common topic with this paper.

When designing a distributed C/S system, there happens to be quite a lot of design decisions that are made at very early stages of development, such as:

- Work distribution between client and server
- Type of servers and clients
- Distribution of functions among servers

Usually the impact of early design decisions on the performance is not very clear. Furthermore, the wrong decisions may result in expensive re-design, re-write of the code or may even waste an entire project.

The idea advocated in [4] is that the design and software performance engineering (SPE) activities can be integrated in an iterative manner. The aim of that approach is to analyze the desing for the purposes of performance and consider alternative design configurations. However, to be able to consider the performance engineering activities at the desing level (i.e. to be able to application and access logic), we need to have the message communication and functionality at the client and server sides.

As an example application a relational databse intensive C/S system is chosen and is observed from the beginning till the end by using the described strategy. As for the design of the project an object oriented approach was chosen, where use cases, structural view (using the object models) and dynamic view (using the collaboration diagrams) of the program was used. The performance modeling of the project was carried out through a language called CLISSPE (Client/Server Software Performance Evaluation). This language was also developed by the authors of the paper and details of the language are given in [3]. As explained in [3], the language was originally developed for the remodelling of a very arge mission-critical system. The use of CLISSPE on this work however, aims at bringing together the design and performance activities by enabling software developers and the performance analysts to work on the use cases collaboratively on the same platform. In more detail, by using CLISSPE designers of the C/S platform specify objects (server, client, database, tables, transactions and networks), the relations between these objects and the transactions executed in the system. Then the compiler of CLISSPE compiles all the design specifications into a queuing network and also the parameters of the system are specified by the compiler so that the performance analyst can work on that network.

I will structure the rest of the paper as follows: In §2 some background information regarding the SPE and the CLISSPE language. In §3 some details of the application that motivated this study will be provided. Then the steps of the proposed model will be given in §4. The analytic models that were utilized by the compiler of the proposed model are provided in §5 and the parameter gathering activities of the reported project are summarized in §6. Finally the reported results will be summarized in §7 and a brief discussion of my own ideas regarding the approach as well as the paper will be presented in §8.

2. BACKGROUND OF CLISSPE

So as to be able to predict the performance of a new systemt that is under development, we need performance models. The model chosen in this paper was queuing networks, which require two parameters:

- workload intensity (e.g. arrival rates)
- demand at each resource

The former of those requirements (workload intensity) can be attained through performance requirements. However, the latter (demand at each resource) is trickier, in the sense that it requires a deep understanding of the domain and the software that is under development. Performance analysts use the knowledge of developers to have an idea of the demands, however, if the software developers are too busy to provide that information at initial stages or if they are unwilling to cooperate with the analysts, then having the demand information proves to be extremely difficult. In the case of this paper, CLISSPE is capable of estimating those demands given that it is provided with objects, mappings and the transactions. CLISSPE is reported to be composed of 3 parts:

1. **Declaration Part:** The following objects are declared in this part: Client, client type, server, server type, disk(s), disk type(s), database management system, database tables, network, network types, transactions, remote procedure calls and constants.
2. **Mapping part:** The objects defined in the previous step are mapped to one another (e.g. clients and servers to networks or DB tables to servers).
3. **Transaction Specification Part:** This section explains the logic of each transaction. For example if it is a loop then the estimate of how many times it is supposed to be executed is given or if it is a branching operation, then the probability of every branch is given.

The details of the language and the specifications of the afore mentioned steps are given in [5]. I have also downloaded and went through the specification document for the language and I have chosen some examples for each step so that we can get a feel of the language as well as its syntax. Below are 3 example statements for each one of the explained steps:

1. **Declaration Example:**

```
# below is the syntax
network_type NetworkType bandwidth= number
          type= { ATM | Ethernet
          | Fast_Ethernet | TokenRing
          | FDDI | WAN
# below is an example
network_type Ring16 bandwidth= 16.0
          type= TokenRing
```

2. **Mapping Example:**

```
# below is the syntax
network NetworkName  type= NetworkType ;
# below is an example
network StationLAN  type= Ring16 ;
```

3. **Transaction Specification Example:**

```
# below is the syntax
transaction TransactionName rate= number ;
# below is an example
transaction assign rate= 0.02
```

3. APPLICATION TO BE MODELED

Since the paper that I am presenting is built upon an application, it is helpful to have a small section that explains this application. The application is reported to be a recruitment and training system (RTS) that is used by a government agency. The application before the renewal was a more than 20 year old mainframe application with a legacy code written in multiple languages. Furthermore, due to its mainframe nature, it has limited capability of scaling.

The renewal process aims at transferring the RTS to a C/S architecture. The new system will consist of several recruitment centers that will be connected with a 10-Mbps ethernet LAN. While the recruitment centers may or may not have a local server for application and database, the headquarters will be aided with multiple application and database servers. Furthermore, the current system uses an old virtual storage access method (VSAM), whereas the new system will be using an ORACLE database.

4. STEPS OF THE MODEL

The proposed model is composed of 8 steps. However, before discussing the steps of the model, some fundamental information regarding the notation and employed modeling language should be provided.

The unified modeling language (UML) used in this paper is used in conjunction with an object oriented analysis and modeling method. This method uses a combination of use cases, object modeling, statecharts and sequence diagrams. The UML notation that was used in the paper is based on UML specified by [1, 2]. The functional requirements of the system are defined by use cases and the actors. The structural (static view of the system) modeling of the system is accomplished via classes as well as class relations, whereas behavioral (dynamic view of the system) modeling is achieved via interaction between use cases. The colaboration of the objects when executing a use case is shown via collaboration diagrams and sequence diagrams. Finally the particular aspects of the system that depend on particular states of the system are shown via statecharts.

Now that I have summarized the used methodology, I will continue with the steps of the proposed iterative, integrated, object oriented method for the desing and analysis of the client and server systems. A sample figure, describing architecture of the steps that are going to be explained in the following subsections is given in Figure 1.

4.1 Step 1: Use Case Definition

In this stage the functional requirements of the system are modeled via use cases. Use case is basically an interaction scenario that defines the interaction between the user/actor and the system[1]. Prior to use cases, the requirements are elicited and when eliciting the requirements, the use case is somewhat considered as a black box. Then the requirements are map to actions and user types. When we analyze separate use cases, then we can see a lot of common actions. Abstract use cases help us reflect the common functionality to different use cases.

A nice example given in the paper for the training system is also included here as Figure 2. Figure 2 shows the main user of the system (personel specialist) and different actions that are available to

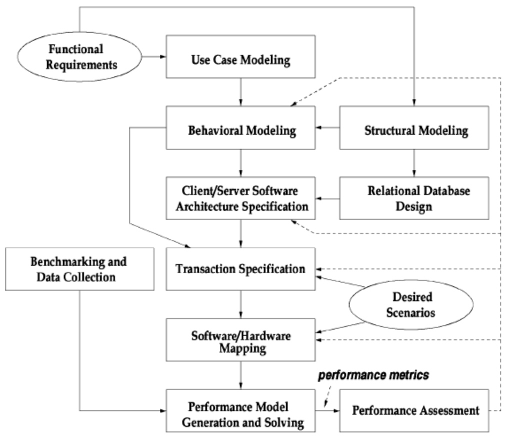
Functional Requirements
Use Case Modeling
Behavioral Modeling
Structural Modeling
Client/Server Software Architecture Specification
Relational Database Design
Benchmarking and Data Collection
Transaction Specification
Desired Scenarios
Software/Hardware Mapping
performance metrics
Performance Model Generation and Solving
Performance Assessment

Figure 1: The architecture of the integrated software design and analysis method for C/S systems.

him for 2 different tasks (checking the skills of a new or an existing employee). In fact there are two concrete use cases here and they use common abstract use cases, which allows us to integrate the two use cases into one as given in Figure 2.

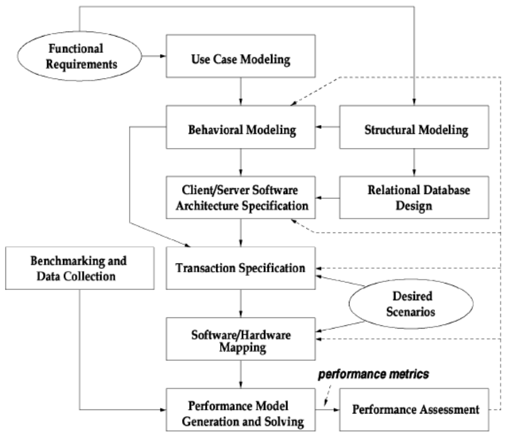
Functional Requirements
Use Case Modeling
Behavioral Modeling
Structural Modeling
Client/Server Software Architecture Specification
Relational Database Design
Benchmarking and Data Collection
Transaction Specification
Desired Scenarios
Software/Hardware Mapping
performance metrics
Performance Model Generation and Solving
Performance Assessment

Figure 2: The use case example of check the skills of an existing employee or a new employee.

4.2 Step 2: Structural Model Definition

The structural model of the system describes the static structure of the system through modeling the real world objects into classes. The structural model defines the classes as well as the attributes of those classes as well as their functions and the interaction between those classes.

The initial consideration of the structural model is the entity classes, which are mapped to database. Entity classes bear particular importance in the sense that they will persist for a very long time in the system and will serve to a number use cases. Another example is the association class, which is used to define the associations between other classes. For example each skill may have sub-skills and each sub-skill may have one or more *"course"* classes that are required for that sub-skill. The structural model of the training system is provided in Figure 3. Note that the static structure of the collaboration diagram will be fundamental when we are designing the relations of the relational database in §4.4.

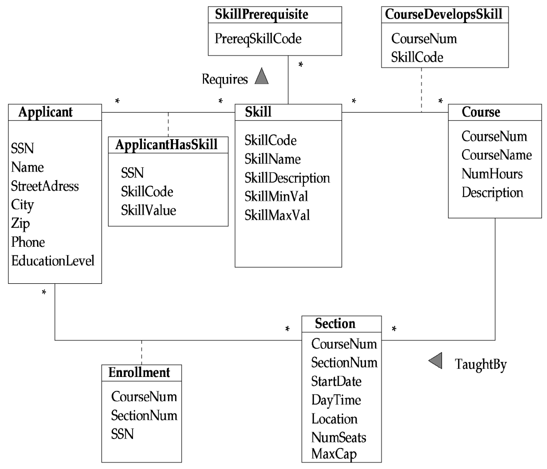

Figure 3: The structural model of the training system.

4.3 Step 3: Behavioral Model Definition

Behavioral model definition tries to model the dynamic behavior of the system. After defining the use cases, the objects from every use case is identified and the message interactions between those objects are defined. The identified sequence of interactions between the objects are defined on an object collaboration diagram.

An example object collaboration diagram (OCD) for the *"check skills"* use case is provided in Figure 4. See in Figure 4 that there is a user interface to the user, which in reality may consist of several different GUI componenets. However, in object collaboration diagram the objects are modeled as application-level objects, hence it will be shown as a single object. Also see from Figure 4 that each OCD has an associated message sequence. This sequence is nothing more than a structured and ordered refinement of the activities that were listed in the use-cases.

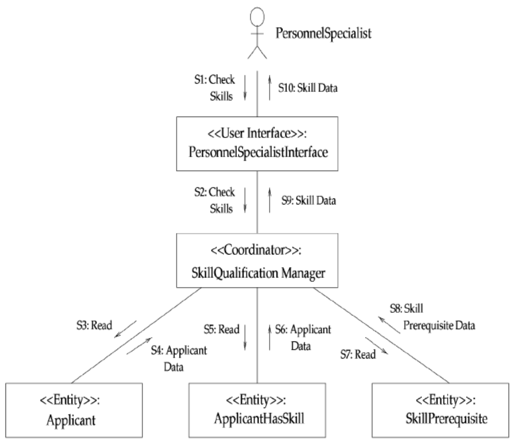
PersonnelSpecialist
S1: Check Skills
S10: Skill Data
<<User Interface>>: PersonnelSpecialistInterface
S2: Check Skills
S9: Skill Data
<<Coordinator>>: SkillQualification Manager
S3: Read
S4: Applicant Data
S5: Read
S6: Applicant Data
S7: Read
S8: Skill Prerequisite Data
<<Entity>>: Applicant
<<Entity>>: ApplicantHasSkill
<<Entity>>: SkillPrerequisite

Figure 4: The object collaboration diagram for the *"check skills"* use case.

4.4 Step 4: Mapping Structural Model to Relational Database

This stage of the method entails the mapping of the objects to particular relations in a relational database. The relations between objects have to do with the static structure of the system, which was identified by the structural model identification. The relations between the classes identified during structural modeling can be summarized as follows:

- Applicant(SSN, name, ...)
- Skill(SkillCode, SkillName,...)
- SkillPrerequisite(SkillCode, PreRegSkillCode)
- Course(CourseNum, CourseName,...)
- Section(CourseNum, SectionNum,...)
- ApplicantHasSkill(SSN, SkillCode, SkillValue,...)
- Enrollment(CourseNum, SectionNum, SSN)

Note that in the above listing, the udnerlined field names are supposed to be the primary keys.

4.5 Step 5: Client Server Software Architecture Development

In this step we will assign the objects that were identified earlier to different C/S architectures. Our aim in this step is going to be able to identify a C/S structure that would provide us different configuration alternatives. Later on these alternatives will be used for different performance evaluations. For example for our problem at hand we can come up with 2 types of C/S configurations:

- 2 Tier Architecture: User interface as well as application layer is provided in the client side, whereas the database is kept in the server node.
- 3 Tier Architecture: User interface is kept in the client node, application functionality is stored in an application server and the third tier becomes the database server, which may be kept in a single node or may be distributed to several different servers.

However, the actual mapping from architecture to system configuration will be done in software and hardware mapping step.

4.6 Step 6: Transaction Specification

The specified transactions in the method defines the business logic of the C/S system. The sequence of transactions that will be written in CLISSPE will be derived from the object collaboration diagram that was given in Figure 4. The specifications of the transactions are coded up using the language of CLISSPE.

Another point that deserves to be mentioned is the fact that there are two parts of a transaction: The client side and the server side. If we are to reference the collaboration diagram of Figure 4, the client part of the transaction corresponds to the user interface object, whereas the server side corresponds to control and entity objects.

A very simple example of a transaction is provided below. This transaction basically shows the skills that a particular applicant is qualified to train for. Note that since we need information from the server side, there is also a remote procedure call (RPC) to the server in that transaction.

```
transaction CheckSkills running_on client
  ! Actor enters applicant SSN
  ! check applicant skills
  rpc check_skills to_server ApplicServer;
  ! Display skills applicant is qualified to
    train for
end_transaction;
```

4.7 Step 7: Hardware/Software Mappings Definition

Once we have defined the system architecture and the transactions, now is the time to map the C/S system architecture components to physical components (such as CPU, network etc.). The architecture components are also given particular characteristics such as latencies and or bandwidth requirements. However, these mappings happen at the language level rather than actually building up the physical system. The CLISSPE language allows the its users to define physical components in the laguage and specify particular characteristics of those physical components. After defining the physical components in the language, they are mapped to one another. For example, server and client machines are tied to a network (note that servers, clients and networks are object defined to represent the physical items). Furthermore, DB tables are assigned to DBMS objects. Below is a simple example for a DBMS system, where server type is set to IBM and the DBMS running on the machine is set to ORACLE with 8,192 KBytes of buffer size and with a dual core.

```
! this goes in the declaration section
server DBServer type= IBM-RS-6000-M43P133
dbms= Oracle DB_BuffSize= 8192 num_CPUs= 2
disk dsk01 type= ServerDisk
disk dsk02 type= ServerDisk;
network_type HQType bandwidth= 100
type= Fast_Ethernet;
network HQLan type= HQType;
```

4.8 Step 8: Performance Modeling and Assessment

In this part of the system all the previously specified system configurations, mappings and transactions are used by the CLISSPE system. The system is composed of the following parts: Specifications, compiler, model parameters, model solver and the model results (throughputs, response times, utilizations). For the ease of understanding Figure 5 is presented. The order of specifications, compiler, model parameters and model solver are presented in that figure.

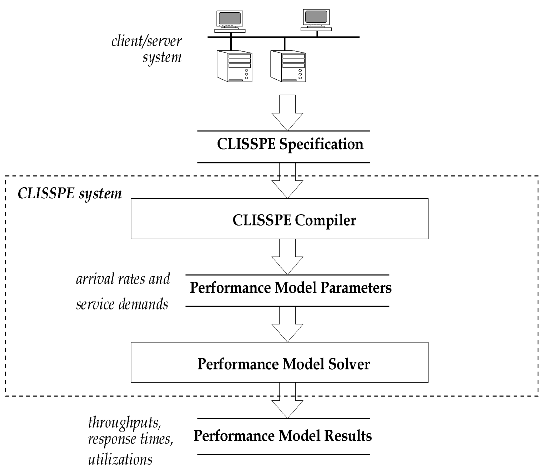
client/server system
CLISSPE Specification
CLISSPE system
CLISSPE Compiler
arrival rates and service demands
Performance Model Parameters
Performance Model Solver
throughputs, response times, utilizations
Performance Model Results

Figure 5: The clisspe system where the components are shown in the order of execution.

Now that we are aware of the organization of the CLISSPE system, I will introduce the notation that we are going to use throughout the rest of the paper to calculate the performance of the system.

After specifying the classes of the system as well as their interactions with one another, the CLISSPE compiler will come up with a multiclass open queuing network (QN). To QN will be represented with $Q = (R, W, \lambda, D)$, where R corresponds to resources, W corresponds to workload, λ corresponds to a vector of arrival rates for transactions for every class and finally D corresponds to a matrix of service demands. The matrix $D = [D_{i,r}]$ corresponds to a $|R| \times |W|$ service demands, where $D_{i,r}$ is the service demand for workload class r at resource i.

4.8.1 *Computing Service Demands*

Since we are given the required notation, in this section we will continue with the calcualtion of service demands in D. Before starting the computation we will define s, which is a statement of a transaction associated with class r. Keep in mind that s may be associated with various CPU and DB demands. The general demand formula is given in Formula 1

$$D_{i,r} = \sum_{s \in S_{i,r}} n_s \times p_s \times D^s_{i,r} \qquad (1)$$

In Formula 1 $S_{i,r}$ is all the statements that contribute to the service demand of of class r at resource i, n_s is the number of times the statement s is executed and p_s is the possibility of that statement being executed. Finally $D^s_{i,r}$ becomes the average service demand at resource i due to single execution of s. The finalization of the values n_s and p_s are influenced by the loops and conditional statements (if, else-if, switch) respectively.

4.8.2 *Modeling the database statements*

When s is a statement database access such as $select$, then the calcualtion of $D^s_{i,r}$ becomes somewhat complicated. In this subsection for each database access statament s the associated number of I/Os, the disk time and CPU times will be calculated. Note that there is a lot of dependency in those calculations depending on how DBMS handles the $select$ statements. For example, the existence of indexes in the DMBS or the size of the buffer as well as the access method to the database (hashing, B-trees) all have an influence on the calculation. For general concepts on the DBMS systems and indexing, access strategies some of the references given in the paper are: [6, 7, 8].

The complication is doubled when we consider the possible access plans that are built on top of these different access methods. Some of the possible access plans are:

- Table space scan, which scans all the rows of a table
- Indexed scan, where one or more indices are present to ease the select statement
  - Single table single index
  - Single table multiple index
  - Two table joins
  - More than two table joins

The assumption made in CLISSPE language while calculating the cost of an access plan is that CPU cost is linearly related to the number of I/O's generated by the access plan. Therefore, the general formula might be summarized as follows:

$$C_{CPU}(AccessPlan) = axN_{I/O}(AccessPlan) + b \qquad (2)$$

In Equation 2, the number of I/O accesses depends on the access plan and the linear relationship of I/O operations to the CPU cost is defined by the constants a and b. The constant b is used for the start-up CPU cost and the constant a stands for the CPU cost per each I/O.

The selection of the optimum access plan for calculating the I/O and CPU costs of a select statement is done by the CLISSPE compiler. By trying out all the possible access plans, the compiler chooses the one with the lowest cost. Note that depending on how the DBMS system was specified in CLISSPE language, the available access plans may change or be restricted.

4.8.3 *Performance Parameter Gathering*

A considerable number of parameters regarding the performance calculations have to be known, before starting the assessment of the new C/S system. The reason for the estimation is that at the desing level, since the system is not operational, those numbers cannot be known. While estimating these numbers the frequency of execution of each use case is estimated. In other words, for our case it is the number of times an existing or a new employee is processed through the system over a certain amount of time.

If the system is a renewal of a legacy system, then the logs of the old system can be adapted to the new system and the frequency of the use cases can be taken as the baseline for the transaction frequencies. So as to test the system for higher workloads, a multiplier greater than 1 can be applied to the frequency elicited from the legacy system.

Not only the frequency of use cases, but also some other parameters such as the execution of loops or the probability of branch statements are needed to be estimated. For those parameters, again the legacy system can be used (if there was retrospective data collection) or some of the domain experts (in the case of no retrospective data) may be utilized. It is reported in the paper that for the particular application on which this particular research was based, all of those methods were employed at different levels. In a similar manner for the size of the dataset, both the investigation of the legacy system as well as interviews were used. CLISPE system allows its users to code up those parameters as constants and probabilities. In the reported system, the authors have come up with 65 constants and 35 probabilities.

4.8.4 *Assessment of the Performance*

As we have noted earlier, the system whose performance analysis was performed in this paper is a recruitment system of a US government agency. Unfortunately, rather than providing all the details of the calculation for the new system, the authors have preferred to include a summary table for the new applicant transaction. This table is comprehensive to some extent, in the sense that it summarizes the response times of the whole system under 8 different configurations. The summary table is given in Figure 6.

So as to be able to understand Figure 6, we first need to explain the notation used. In Figure 6 there are 8 scenarios. Each scenarios is represented one of the two forms: $mD(p)nA(q)$ or $Comb(p)$. In $mD(p)nA(q)$, m is the number of database servers in the scenario and p is the number of processors in every database server, whereas n is the application server number and q is the number of pre-processors in the application servers. In $Comb(p)$ the application and the database servers are on the same machine that has p number of processors. Lastly, notice that some of the scenarios have a suffix of SC, which stands for selective caching, i.e. some of the database tables are stored entirely in the main memory.

5. ANALYTIC MODELS FOR COMPILER

6. PARAMETER GATHERING

	Load Multiplier			
	1.0	1.5	2.0	2.5
Arrival Rate (tps)	0.047	0.071	0.095	0.12
Scenario	Response Time (sec)			
1D(2)1A(1)	46.7	83.2	4243	N/A
1D(2)1A(1)SC	15.7	22.8	3436	N/A
3D(2)1A(1)	30.8	37.0	48.2	76.4
3D(2)2A(1)SC	9.5	10.1	10.9	12.3
1D(4)1A(2)SC	8.0	8.4	9.1	10.3
1D(3)1A(1)SC	8.0	8.5	9.2	10.4
Comb(6)SC	5.8	5.8	5.8	5.9
Comb(4)SC	5.8	5.8	5.8	5.9

Figure 6: The response times for the transaction: Check new applicant.

7. RESULTS

8. DISCUSSION

9. REFERENCES

[1] G. Booch, J. Rumbaugh, and I. Jacobson. *Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005.

[2] H. Gomaa. Designing concurrent, distributed, and real-time applications with uml. In *Proceedings of the 28th international conference on Software engineering*, ICSE '06, pages 1059–1060, New York, NY, USA, 2006. ACM.

[3] D. A. Menascé. A framework for software performance engineering of client/server systems. In *Proc. 1997 Computer Measurement Group Conf.*, 1997.

[4] D. A. Menascé and H. Gomaa. A method for design and performance modeling of client/server systems. *IEEE Trans. Softw. Eng.*, 26:1066–1085, November 2000.

[5] D. MenascÃľ. Clisspe: A language for client/server software performance engineering. Technical report, Dept. of Computer Science, George Mason University, 1997.

[6] A. Swami and K. B. Schiefer. Estimating page fetches for index scans with finite lru buffers. *SIGMOD Rec.*, 23:173–184, May 1994.

[7] C. M. Woodside, J. E. Neilson, D. C. Petriu, , and S. Majumdar. The stochastic rendezvous network model for performance of synchronous client-server-like distributed software. *IEEE Transactions on Computers*, 44:20–34, 1995.

[8] S. B. Yao. Optimization of query evaluation algorithms. *ACM Trans. Database Syst.*, 4:133–155, June 1979.