A Regression-based Method for Software Performance Engineering

Omid Bushehrian
Department of Computer Engineering and Information Technology, Shiraz University of Technology, Iran

Reza Ghanbari Baghnavi
Department of Computer Engineering and Information Technology, Shiraz University of Technology, Iran

ABSTRACT
In this paper a statistical methodology for finding the optimal deployment of distributed software objects over computational nodes is presented. The optimal placement of a distributed software objects, from the performance viewpoint, has a significant impact on the performance of the software. In the proposed methodology, a performance predictor function is extracted from a dataset of simulation results using the regression analysis. This performance predictor function then is used by an optimization algorithm to find the optimal object deployment. The key advantage of the proposed methodology over using the traditional QN models is that solving the predictor model obtained from the QN approach during the optimization process many times, particularly when the search space is huge, is prohibiting due to its time complexity.

Keywords: Software Performance Engineering; optimal object deployment, simulation, Finite State Process

1. INTRODUCTION
Quantitative software performance evaluation is a common practice during the early stages of the software development aiming at satisfying QoS constraints such as response time[17][14]. In distributed software, the optimized assignment of objects and components over the computational nodes has a significant impact on the software performance. For example, assignment of two communicating objects on the same computing node eliminates the network delay caused by message passing between them while results in some computational delay due to increasing the computational load on that node. Finding the deployment of objects which results in the lowest response time of the software scenarios is an optimization problem and cannot be performed manually. Automated tools usually apply heuristic search methods to explore the search space and evaluate each deployment within the search space from the performance viewpoint[3][13][7]. Therefore an analytical predictor model(function) is required for fast evaluation of each deployment during the search. The main shortcoming of using conventional Queuing Network models in this optimization problem is the fact that automatic generation and solving the multi-layer QN models corresponding to each deployment during the search time is very time consuming and complex. The main reason for this costly model solution stems from the fact that the resulting multi-layer QN are not product-form and therefore the approximation algorithms have to be used for their solution which are inherently iterative[14].
In this paper a regression analysis method is applied to extract a performance predictor function, from a dataset of simulation records, which can be solved very fast either when using a heuristic or Linear Programming optimization method.

In our previous work[6], we studied the effect of input work load to a use-case scenario, on the optimal deployment of objects collaborating in that scenario. To illustrate this effect a simple example is presented in Figure 1. There are two nodes, two objects: *server* and *worker* and two resources: a *high speed link* and a *database*. The *server* object need to acquire the *high speed link* resource which is associated with $Node_1$ and the *worker* object requires some service form *database* which is associated with $Node_2$. Besides, the *worker* object provides service to the *server* object and therefore there is a dependency between them. Assume that the service demands corresponding to the server and the worker objects, denoted by D_s and D_w respectively, are D_s=1 ms and D_w=5 ms. First, the system is lightly loaded with input workload equal to 150 request per second($workload_1$=150 req/s). Therefore according to the utilization Law [14] the values of *server* and *worker* utilizations will be: $U_s = D_s \times X_o = 1 \times 0.15 = 15\%$ and $U_w = D_w \times X_o = 5 \times 0.15 = 75\%$, where X_0 is the overall throughput of the software. Therefore at this workload, the total utilization of objects is less than 100% and hence the placement of two objects on the same node has no computational delay(the delay caused by sharing the same node with several objects) (Figure 1 top). By increasing the workload to 200 req/sec, the utilization of *server* and *worker* objects will be 20% and 100% respectively. Assuming that each node has one CPU installed on it, at this workload the total utilization of objects exceeds 100% and therefore some computational delay is produced. By moving the *worker* object to $Node_2$, this computational delay can be eliminated (Figure 1 bottom).

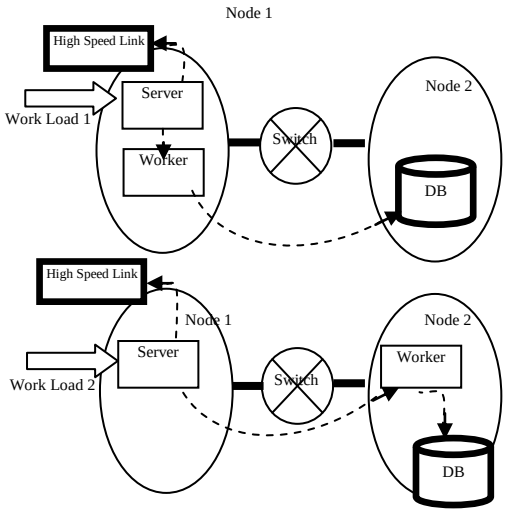

Figure 1. Two different deployments of objects. By increasing the workload(load2>load1) moving the Worker object to Node2 reduces the computational delay on Node1.

The important result from this example is the fact that, there is no unique optimal deployment of objects; rather, corresponding to each workload to the software, a different deployment of objects will be optimal.

2. RELATED WORKS

Most researches in the field of performance engineering are dedicated to the creating and analysis performance models for an existing software deployment or configuration rather than automatic optimization of them.

In[9], for optimizing the performance of a software in the presence of simulation results a *Response Surface Method* is presented and its usefulness for optimal capacity planning is analyzed. However, the applicability of this method for more complex optimization problems such as object deployment has not been studied. One of the earliest works in the field of deployment optimization is presented in [16]. In this paper, a static task allocation method is proposed. However, the effect of input workload of the use-case scenario in determination of the optimal task allocation is not studied in this paper. In [3] two optimization methods for the problem of object deployment are used and compared: Binary Integer Programming and Genetic algorithm. However in this paper only the minimization of the communicational delay between components is considered (by minimizing the number of transmitted messages between components). In [5] a Linear Integer Programming-based method for the problem of finding the optimal deployment of components is used. In this paper the optimization objective is to find a deployment for which the computing and network delays are minimized. In [10] the optimal deployment of objects is obtained by using a partitioning algorithm. This algorithm partitions the object graph of the software such that the objects with most communication are located on the same machine. [1] is another paper that considers only minimization of the "communicational delay" during the optimization of the software architectural models using evolutionary methods and ignores the concept of "computational delay"[6]. A multi-criteria objective function is presented in this paper that evaluates the data transmission reliability and communication overhead of a deployment.

The most related work to ours is presented in [13]. In this paper a multi-criteria genetic algorithm for optimizing the software architecture for performance, cost and reliability is presented. The genetic algorithm in this work evaluates different architectural alternatives in the search space during the generations, to find the one for which the value of objective function is minimized. The evaluation of each solution (architecture) within the population from the performance viewpoint is performed by automatically generating a Layered Queuing Network model corresponding to the solution and analyzing it at runtime. However it is a very time consuming and complex method particularly for softwares with many components. On the contrary, our optimization model is designed to predict the fitness of each solution very fast.

In overall, most previous works in the area of deployment optimization problem either ignore the computational delay resulted from placing the objects on the same machine in their methods or ignore the fact that the input rate to the system directly affect the optimal deployment of objects. In this paper these two factors are considered in the proposed predictor model.

3. THE REGRESSION BASED METHODOLOGY

Figure 2 depicts the main steps of our regression-based methodology for finding the optimal deployment of objects collaborating in a use-case scenario u. The objective of this methodology is to find the predictor function P(d, λ) that estimates the response time of the use-case scenario u when the inter-arrival time of the successive requests to this scenario follows an exponential distribution with expected value λ and the object distribution is like *d*. To achieve this a dataset consisting of tuples: (d, λ, R) is collected by simulation in which R is the response time corresponding to the deployment d when the input load is λ(steps 1-5 in Figure 2). This data set then is analyzed to find the predictor function P(d, λ)(step 6). The next step is to use this function as the objective function of an optimization algorithm to find the optimal deployment d_{opt} corresponding to a given input workload λ for which the value of $P(d_{opt}, \lambda)$ is minimum(steps 7-8).

In the following subsections the steps presented in Figure 2 are explained in detail. Before that, a case-study adopted from[7] is presented here. This is a web application used in ISP companies by which a customer can purchase an account for accessing Internet. The main success scenario of the Purchase-Account use-case is as follows:

"(1)The customer navigates to the purchase page. (2) The system shows available Internet packages. (3) The customer selects a package and confirms. (4) The system navigates to the bank payment page. (5) The customer pays the package price. (6) The system verifies the payment and creates an account in AAA software and submits the created account to the customer."

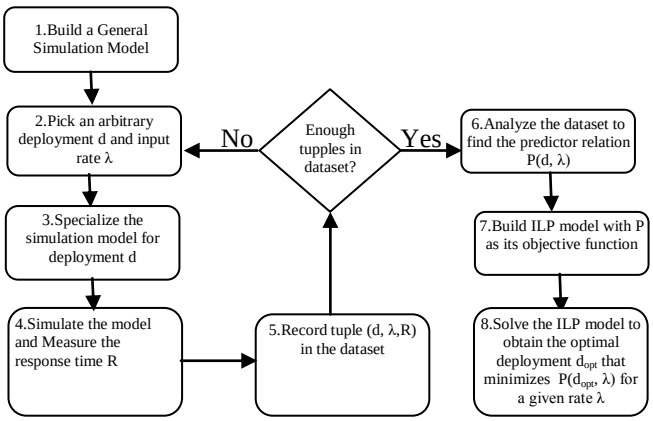

Figure 2. The regression based verification steps

In this case study, in addition to the fact that the purchase requests to the application is often high, the *Accounting* object (in Figure 3) which provides access to the AAA(Authentication, Authorization and Accounting) software is very busy due to processing of many authentication requests from different Access Points inside the company network. Therefore it is observed that without the correct deployment of objects over the available resources many user requests to the system may fail at step (6) of the scenario. Therefore finding the optimal deployment of objects corresponding to each input rate to the system is crucial. The communication diagrams corresponding to this scenario is shown in Figure 3. Note that objects with the <<*resource*>> stereo-type represent resources and have fixed location.

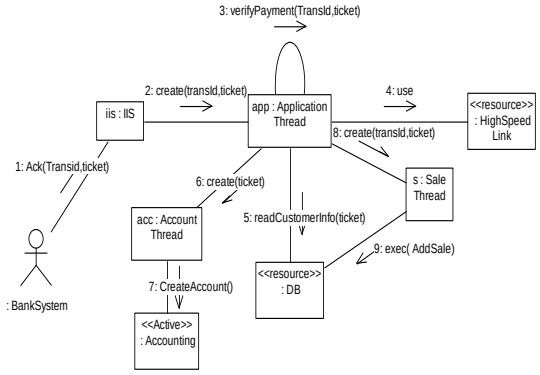

Figure 3. *Purchase-account* scenario

Table 1. The performance attributes for communication diagram presented on Figure 3

Method	Description	Demand(exponentially distributed)
iis.Ack()	OpenWorkLoad(exponentially distributed)	
app.Create()	Creates a new Application Thread	15 ms
app.VerifyPayment()	Uses the high speed link associated with Node1	60 ms
DB.readCustomerInfo()	Reades the customer information from DB	55 ms
acc.Create()	Creates the accounting thread	10 ms
Accounting.CreateAccount()	Creates an account in AAA server	90ms
s.Create()	Creates the sale thread	10 ms
DB.Exec()	Records the current Sale in the DB	55 ms

3.1 Building the General Simulation Model

At the first step of the algorithm, a *general simulation model* for the scenario is built using the FSP language [4][12] considering the service demands of the objects presented in Table 1. By "*general"* we mean that, this model is deployment independent and only considers objects and their behaviors. An FSP model is a collection of concurrently executing *processes*. Each process is indeed an abstract state machine which performs some *actions* consecutively [12]. For transforming sequence or communication diagrams into an FSP model, first we must transform each object to a *process*.An algorithm for building an FSP model from a scenario is presented in our previous paper [8]. Therefore, in the simulation model, a *process* is defined corresponding to each active object in the scenario like *iis*, *DB* and *Accounting* (see Figure 4). Each object has to acquire the CPU time on which it is deployed, when it is about to perform some computation and then it has to release the CPU when the computation is finished. To model this behavior, each process emits a *getcpu* action prior to starting its computation and a *freecpu* action after the computation is finished. These actions are later synchronized with the corresponding actions of the computing nodes on which they are deployed when the model is specialized for a specific deployment (step 3 of the algorithm in Figure 2).

IIS = (request ->**iis_getcpu** -><?exp(1.0/15)?> createApp->**iis_freecpu**->**iis_getlink1**-><?exp(1.0/7)?>**iis_freelink1**->IIS).

DB=(getdb->**db_getcpu**-><?exp(1.0/55)?>freedb->**db_freecpu**->DB).

Accounting=(getAcc->**accounting_getcpu**-><?exp(1.0/90)?>finAcc->**accounting_freecpu**->Accounting).

Figure 4. The processes corresponding to active objects

In addition to CPU, an object may also require a link to perform a remote invocation to another object. Therefore, in the corresponding process, prior to the remote invocation I_i, the process has to emit a $getlink_i$ action to acquire the link and after completing the invocation it has to emit the $freelink_i$ to release the link. These actions are later synchronized with the corresponding actions of the links when the model is specialized for a specific deployment (step 3 of the algorithm in Figure 2).

In the communication diagram of the *purchase-account* scenario, there are also three objects of type thread: *app*, *s* and *acc* which should be modeled by the FSP language. In order to model a multithread object in the FSP language, the method presented in [8] is used. A multithreaded object *objT* is defined as a process in FSP with T instances:

$$[0..T]:objT$$

Where, T is the size of thread pool corresponding to *objT*. By this definition we say that there are T available threads for object *objT*. In our model we chose T=100. To model creation and starting a thread by the parent object *objP*, the *createThread* action in the process *objP* has to be synchronized with the starting actions of T instances of *objT*:

$$[0..T].start/createThread$$

Where, *start* is the starting action of the *objT* process. To model the computing nodes (each node has one CPU installed on it) and network links, corresponding to each one a process is defined in the FSP model as shown in Figure 5. In this Figure, $L_{i,j}$ denotes the link between node i and node j.

N_1=(getnode1->freenode1->N1).
N_2=(getnode2->freenode2->N2).
N_3=(getnode3->freenode3->N3).
N_4=(getnode4->freenode4->N4).

$L_{1,2}$=(get12->free12->L12).
$L_{1,3}$=(get13->free13->L13).
$L_{1,4}$=(get14->free14->L14).
$L_{2,3}$=(get23->free23->L23).
$L_{2,4}$=(get24->free24->L24).
$L_{3,4}$=(get34->free34->L34).

Figure 5. The processes corresponding to computing nodes and network links

The actions of the processes corresponding to the nodes and links in Figure 5 should be synchronized with the correct actions of the processes which are deployed on them. This step is performed when the model is specialized for a specific deployment.

3.2 The Simulation Model Specialization

Model specialization is the process of altering the general FSP model, built for *purchase-account* scenario, according to a specific object deployment *d*. To do this, the *getcpu*, *freecpu*, *getlink* and *freelink* actions of the processes representing the scenario objects, have to be synchronized with the

corresponding actions of the computing nodes and links according to deployment d. For example consider the deployment in which *iis* and *app* objects are located on node number 1, *DB*, *Accounting* and *s* objects are located on node number 4 and *acc* is located on node number 2. The required synchronizations to specialize the general model for this deployment are presented in Figure 6.

Similar statements have to be added to the specialized model for synchronizing the *getlink* and *freelink* actions of the underlying link processes to the upper layer remote invocations.

```
db_getcpu/getnode4,   //DB   fixed
db_freecpu/freenode4,  // DB fixed

accounting_getcpu/getnode4,   // Accounting
accounting_freecpu/freenode4,  //Accounting

iis_getcpu/getnode1,   //iis
iis_freecpu/freenode1,  //iis

[1..100].acc_getcpu/getnode2,   //acc Thread
[1..100].acc_freecpu/freenode2, //acc Thread

[1..100].s_getcpu/getnode4,     //sale Thread
[1..100].s_freecpu/freenode4,  //sale Thread

[1..100].app_getcpu/getnode1,     //app Thread
[1..100].app_freecpu/freenode1,  //app Thread
```

Figure 6. Action synchronization for a given deployment in FSP language

3.3 Building the Dataset

After specializing the simulation model for an arbitrary deployment *d*, the resulting model is simulated using the LTSA tool [2] to measure the response time of the scenario for different input loads. This experiment was repeated five times for request inter-arrival times: 50, 100, 150, 200, 300ms and for 30 different deployments ended up with a dataset comprising 150 tuples. Since the general model is built once the experiments are performed very fast. In order to add a tuple (d, λ, R) to the dataset, deployment *d* should be represented by numerical variables.

In this case-study, each deployment *d* itself is represented by four decimal variables (n_1,n_2,n_3,n_4) corresponding to four computing nodes N_1 toN_4. To calculate the values of these four variables for a given deployment *d*, a vector v_i is assumed corresponding to each variable n_i(i=1..4). The number of elements in each vector v_i is equal to the number of objects in the case-study, which is seven, and the elements are used to store binary values with default value 0. To compute the decimal values of variables n_i(i=1..4) corresponding to deployment *d*, the j^{th} element of the vector v_i is set to 1 if object o_j is deployed on node N_i, otherwise it remains 0. Therefore, corresponding to each variable n_i(i=1..4) a binary string v_i is obtained. By converting this binary string to its equivalent decimal number, the value of n_i is obtained. For example consider the following deployment of objects in this case-study (Figure 3):

o_1: *iis* is deployed on node N_2

o_2: *app* is deployed on node N_3

o_3: *HighSpeedLink* is deployed on $Node_1$

o_4: *acc* is deployed on $Node_1$

o_5: *Accounting* is deployed on $Node_1$

o_6 : *s* is deployed on $Node_2$

o_7: *DB* is deployed on $Node_4$

The element values of vectors v_i(i=1..4) are set as follows:

$v_{1=}0011100= (28)_{10}$

$v_2=1000010=(66)_{10}$

$v_3=0100000=(32)_{10}$

$v_4=0000001=(1)_{10}$

Therefore, the values of variables (n_1,n_2,n_3,n_4) corresponding to this deployment is (28,66,32,1). These values along with the values of input rate and response time, are added to the dataset as a new tuple. A small part of resulting dataset is shown in Table 2.

Table 2- Sample tuples in the collected dataset

n1	n2	n3	n4	Inter-arrival rate(ms)	Response time
80	36	8	3	150	562
16	68	40	3	150	522
112	2	8	5	150	495
80	32	8	7	150	462

3.4 Analyzing the Dataset by SAS

By analyzing the dataset by the SAS tool[15], a significant association between the value of response time: RT of a deployment and the independent variables n_1,$n_2$$n_3$,$n_4$ and *r* was found (P<0.01):

$$RT = 843 - .8 * n1 - .3 * n2 + .2 * n3 + .7 * n4 - 1.99 * r \quad (1)$$

3.5 Building the ILP model

To find the deployment which minimizes relation (1) for a specific input rate value, we used the Integer Linear Programming (ILP) method with the following constraint:

$$\sum_{o \in O} \sum_{n=1}^{n=4} So, n = 1 \qquad (2)$$

Where $S_{o,n}$ is a decision variable. The value of $S_{o,n}$ equals 1 when the object *o* is deployed on node *n* , otherwise it is 0. The objective of the ILP model is to minimize the value of RT in relation (1). Therefore we used relation (1) as the objective function of the ILP model reminding that the value of independent variables n_i should be computed in terms of decision variables $S_{o,n}$ as follows:

$$n_i = \sum_{j=0}^{j=6} v_i[j] \times 2^j \qquad (3)$$

Where v_i[j] is determined by the value of decision variable $S_{o,n}$ as explained earlier. The relation between the R variable and each of the independent variables in the dataset is assumed to be linear (see relation 1). We obtained the optimal deployment by solving the model presented using LINGO [11] solver which is a tool for solving Linear, Nonlinear, Quadratic and Integer optimization models.

4. APPROXIMATION ERROR RATES

As explained before, for each input workload λ to the system an optimal deployment d_λ corresponding to the workload λ is

obtained by the proposed regression-based model. To show that d_λ is the optimal deployment corresponding to workload value λ with a good approximation we calculated the relative error corresponding to the predicted response times by the regression model. To achieve this, a simulation approach was chosen. An FSP model for deployment d_λ was generated. This model then was simulated using the LTSA and the resulting response times of this model were recorded. The simulation and predicted results for optimal deployments d_λ are presented in Table 3. It is observed that the relative error in our experiments is less than 2.1%. In Table 3, the optimal deployment corresponding to inter-arrival rate λ is represented by a string of seven numbers in which the i^{th} number indicates the node number on which object o_i is deployed. (Objects $o_1$to o_7 in our case study are as follows: o_1: *iis*, o_2: *app*,o_3: *HighSpeedLink* ,o_4: *acc* ,o_5: *Accounting* ,o_6 : *s* ,o_7: *DB*)

Table 3- The optimal deployments and their response times

Optimal Deployment(d_λ)	Inter-arrival rate(ms)	Predicted-Response time	Measured Response time(simulation)	Relative error(%)
2,3,2,1,2,4,1	50	718.9	730	1.5
1,4,2,3,4,4,1	100	615.4	629	2.1
3,3,4,2,1,4,1	300	271.4	276	1.6

5. CONCLUSIONS AND FUTURE WORKS

In this study, a regression-based methodology for the fast evaluation of different deployments of objects collaborated in a software scenario from the performance perspective, is presented. Since generation of conventional performance models such as QN models corresponding to each object deployment is a complex and time consuming task at runtime, the presented model can be applied as an approximate performance estimator in place of QN models in the deployment optimization problems particularly when the search space is huge.

As the future work for this research, we are trying to verify the model by using larger software with more objects to be deployed.

6. REFERENCES

[1] Aleti, A., Bjornander, S., Grunske L., & Meedeniya, I.(2009). Archeopterix: An extendable tool for architecture optimization of AADL models, International workshop on Model-based Methodologies for Pervasive and Embeded Software (MOMPES)

[2] Ayles, T., Field, A.J., & Magee, J.N.(2003).Adding performance evaluation to the LTSA tool, Proc. 13th Int. Conference on Computer Performance Evaluation: Modeling Techniques and Tools, Lecture Notes in Computer Science, LNCS 2794: Springer

[3] Bastarrica, M., Caballero, R., Demurjian, A., & Shvartsman, A.(2001). Two optimization techniques for component-based systems deployment, Proc. 13th Int. Conference on Software Eng. and Knowledge Eng: (SEKE2001)

[4] Bennett, A., & Field, J.(2004).Performance engineering with the UML profile for schedulability, performance and time: a Case-Study, Proc. 12th annual Int. Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems: IEEE

[5] Boone, B., Truck, F., & Dhoedt, B.(2008). Automated deployment of distributed software components with fault tolerance guarantees, Proc. 6th Int. Conference on Software Engineering Research, Management and Applications: IEEE

[6] Bushehrian, O., & Ghanbari, R.(2011). An INLP Approach for Simulated-Based Automatic Objet Deployment", The 2011 IEEE Int. Symp. On Computer Science and Software Engineeirng, Tehran, Iran:IEEE

[7] Bushehrian, O.(2011). The Application of FSP Models in Automatic Optimization of Software Deployment, 18th Inte. Conf. on Analytical and Stochastic Modelinf Techniques and Applications(ASMTA 2011), Lecture Notes in Computer Science, LNCS 6751, Venice, Italy:Springer

[8] Bushehrian, O.,& Ghaedi, H.(2011). The Application of FSP Models in Software Performance Engineering: A Multi-Threaded Case-Study, Symposium on Computers and Informatics(ISCI 2011), Malaysia: IEEE

[9] Chih-Chieh, H., & Devetsikiotis, M.(2007). An Automatic Framework for Efficient Software Performance Evaluation and Optimization, 40th Annual Simulation Symposium (ANSS '07), USA: IEEE

[10] Deb, D., Fuad, M., & Oudshoom, M.J.(2006).Towards autonomic distribution of existing object oriented programs, Proc. of Conference on Autonomic and Autonomous Systems (ICAS06): IEEE

[11] LINGO Users Guide, http://www.lindo.com

[12] Magee, J., & Kramer, J.(1999). Concurrency:State Models and Java Programs, Chichester, England: John Wiley and Sons

[13] Martens, A., Koziolek, H., Becker, S., & Reussner, R.(2010). Automatically Improve Software Architecture Models for Performance, Reliability and Cost Using Evolutionary Algorithms, Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering, USA

[14] Menasce, D.A., Almeida, V.A.F., & Dowdy, L.W.(2004). Performance by Design: Computer Capacity Planning by Example , ISBN 0-13-090673-5: Prentice Hall PTR.

[15] SAS Users Guid, http://www.sas.com

[16] Woodside, M.,& Monforton, G.(1993).Fast Allocation of Processes in Distributed and Parallel Systems, IEEE Trans. On Parallel and Distributed Sys., vol. 4, pp. 164-174:IEEE

[17] Woodside ,M., Franks, G.,& Petriu D.(2007). The future of software performance engineering, Int. Conference on Software Engineering: IEEE.