An Investigation on Estimating Demand Data and Semantic Resource Allocation

S.Ranjithkumar [#1], Dr J.Selvakumar [*2]

[#1]PG Scholar ME-Software Engineering Final yea, Sri Ramakrishna Engineering College Coimbatore, Tamil Nadu, India

*[*2] Associated Professor,*
Sri Ramakrishna Engineering College Coimbatore, Tamil Nadu, India.

ABSTRACT— The objective of Software Engineering is to develop software product effectively. Software services are too complex and it has many capabilities, each corresponding to a business level concept. Customer requires a service that exploits only a fraction of the service's capabilities. Each capability uses many different software functions that cause demands on distributed or multitier set of resources such as CPUs.

The results of these predictions will help the schedulers to improve the allocation of resources to the different tasks. The technique is used to support system sizing and capacity planning exercises, costing and pricing exercises, and to predict the impact of changes to a service upon different service customers. In this paper, we present a framework which uses semantically enhanced historical data for predicting the behavior of tasks and resources in the system, and allocating the resources according to these predictions.

Keywords — **Benchmarking, Linearity, Multicollinearity, Resource Demand Estimation, Statistical Regression.**

I. Introduction

When running a small business, it is important to have an idea of what you should expect in the way of sales. To estimate how many sales a company will make, demand estimation is a process that is commonly used. With demand estimation, a company can gauge how much to produce and make other important decisions. Demand estimation is a process that involves coming up with an estimate of the amount of demand for a product or service. The estimate of demand is typically confined to a particular period of time, such as a month, quarter or year. While this is definitely not a way to predict the future for your business, it can be used to come up with fairly accurate estimates if the assumptions made are correct.

In the absence of centralized control, such behaviors' may be self-reinforcing, leading to a rapid degradation of system performance. A major challenge within open markets is the ability to satisfy service demand with an adequate supply of service providers, especially when such demand may be volatile due to changing requirements or fluctuations in the availability of services. Ideally, this supply and demand should be balanced; however, when consumer demand changes over time, and providers independently choose which services they provide, a coordination problem known as 'herding' can arise bringing instability to the market. This behavior can emerge when consumers share similar preferences for the same providers, and thus compete for the same resources. Likewise, providers which share estimates of fluctuating demand may respond in unison, withdrawing some services to introduce others, and thus oscillate the available supply around some ideal equilibrium. One approach to avoid this unstable behavior is to limit the flow of information between agents, such that they possess an incomplete and subjective view of the local service availability. A model called Demand Estimation with Confidence (DEC) requires the preparation and execution of a number of benchmarks under controlled conditions. A benchmark submits a semantically correct sequence of requests to a system under study. Resource demands are measured for each benchmark separately.

DEC enables a measurement-based test that can be used to validate demand estimates as well as ascertain whether a given service would achieve its performance requirements when executed on a target platform. Together these characteristics of DEC can help service providers assess the risks of providing services to new customers based on customer-specific workloads

II. METHODOLOGY

A. DEC-BASIC Method

This section provides a brief summary of the method used within DEC-BASIC to compute a linear combination vector L for a subset of benchmarks. As stated previously, demand prediction problem a ratio computation technique that devised previously to support synthetic workload generation. Although the modifications required relative to the previous work were fairly straightforward, the technique described here for the sake of clarity and completeness. Let the product FX, correspond to a customer's desired use of a system's functions. The problem of computing L is to determine a linear combination of a subset of the B benchmarks that results in FX. In other words, let A^* be a M×K matrix containing function execution counts vectors for K benchmarks, $K \leq B$, upon the M functions. The K rows of L represent benchmark execution counts for the benchmarks corresponding to the function counts vectors of A^*. If the chosen benchmarks were run with these benchmark execution counts, the resulting function counts would be FX. The problem of computing the linear combination vector L is to determine an $A^*$and L such that the following conditions are satisfied:

$$A^*L = FX, \qquad L(l) \geq 0 \ \forall l. \tag{1}$$

Above equation specifies that the function counts achieved by combining the benchmarks in A^* according to the benchmark counts in L should equal the desired function counts given by FX. Equation restricts the computed benchmark counts to be nonnegative values. To solve the problem, an algorithm is devised which iteratively determines the A^* matrix and the L vector that satisfy the conditions given. Fig provides a high-level overview of the algorithm. The steps of the algorithm are as follows:

1. To begin, an initial A^* matrix is determined by identifying a small subset of the B benchmarks. This relies on the computation of an algebraic basis set for the B benchmarks. The algebraic basis set has the property that a benchmark not belonging to the set can be synthesized as a linear combination of benchmarks in the set with respect to its function execution counts. In effect, the basis set contains benchmarks that are distinct from one another in terms of their function execution counts. MATLAB rref function used to compute the algebraic basis.

2. At each step of the iteration of DEC-BASIC, linear programming is used to find a value of L for a given A^* such that the difference between the desired function counts FX and the achieved function counts A^*L is minimized. Equation forms one of the constraints of the LP problem. The second constraint is obtained by relaxing the condition specified to that given by

$$A^*L \leq FX. \tag{2}$$

This change facilitates an iterative solution by which A^* is progressively modified by adding more benchmarks to the initial basis set until an L that satisfies the stricter constraint given, i.e., corresponding to a better match between FX and A^*L is found. MATLAB linprog function is used for solving the LP problem. The detailed LP formulation is presented in the Appendix.

3. The difference between the desired function counts and the achieved function counts is calculated as an M1 slack vector E ¼ FX * A^*L. The algorithm terminates if the mismatches in function execution counts (given by the elements of E) are less than or equal to user-specified mismatch thresholds for function execution counts. By default, as shown in Fig, the mismatch thresholds are zero signifying a desire to achieve an exact match of FX. However, thresholds can be specified to indicate that certain functions, e.g., those that are resource intensive, need to be matched more closely than others. The algorithm also terminates if a user-specified maximum number of iterations is reached or if all available benchmarks have been used to synthesize FX.

4. If the algorithm does not terminate in step 3, then the benchmark that offsets E the most is identified from the remaining benchmarks in B that are not part of A^*. The new benchmark to be selected is determined by computing the Euclidean distances between E and the function execution counts vectors of the remaining benchmarks. The function execution counts vector that yields the minimum distance is selected and appended to A^* as an additional column. This is followed by another iteration of the algorithm. The algorithm is guaranteed to terminate when the goal is to match a workload mix of functions that results from the B benchmarks. This is because, in the worst case, all B benchmarks will be included to achieve the match. When an exact match of mix is not possible with a given set of benchmarks, DEC can be instructed to match certain functions, e.g., resource intensive functions, more closely than other functions. Specifically, by relaxing the requirements for a certain set of functions one can obtain more exact matches for the other functions not in this set. The next section describes a new method, named DECITERATIVE, which automates such a process.

DEC-BASIC ALGORITHM

1. Create algebraic basis set of k benchmarks from B available benchmarks.
2. Form A* matrix in dimension M x K corresponding to basis set.
3. The difference between the desired function counts and the achieved function counts is calculated to solve LP problem to obtain L that minimizes F X – A*L.
4. Check A*L= = FX or K= = B
5. The algorithm terminates if the mismatches in function execution counts are less than or equal to user-specified mismatch thresholds for function execution counts.
6. If the algorithm does not terminate then select a benchmark which is not in the basis set with best offsets F X – A*L.

7. Add selected benchmark to basis set and form A* matrix of the new set.
8. After that reflect the size of new set. Then increase the value of k by 1
9. Then repeat the process to solve LP problem by Go to step:5

B. DEC-ITERATIVE Method

The main idea behind DEC-ITERATIVE is to aggregate functions together when an exact match of mix is not possible. Several different criteria could be used to decide which functions need to be aggregated. Aggregating functions based on their impact on system performance. Specifically, functions are ranked based on performance impact. This can, for example, be achieved by observing the per-function mean response times for the M functions when the suite of B benchmarks is executed against the system under study. Alternatively, functions could be ranked based on the per-function demands as estimated by regression. The ranking is based on response times for the following discussion. Let the function with the highest mean response time be assigned a rank of 1 while the function with the lowest mean response time is assigned a rank of M. When an exact match is not possible, the two least resource intensive, i.e., least ranked functions are combined together. The execution count for this combined function is set to be the sum of the execution counts of its two component functions.

DECBASIC is then invoked to match for a mix expressed in terms of the counts for the non-aggregated functions and the count for the newly combined function. The process of aggregation is repeated if an exact match of this mix is not achieved. With such an approach the dimension of the mix is iteratively reduced by 1 from its initial value of M till DEC-BASIC reports exact matches with respect to the re dimensioned mix or till M - 1 function have been aggregated. In effect, the aggregation process iteratively lumps functions with similar demands together starting from the least resource intensive functions. Fig. provides a more formal overview of the process. Assume that G is an M - 1 matrix obtained by sorting FX according to the performance impact of the M functions. Specifically, the ith row of G represents the function execution counts of the ith most resource intensive function. Let A be an M ×B matrix containing function execution counts vectors for the B available benchmarks upon the M functions such that A_{ij} represents the number of times benchmark i invokes the j^{th} most resource intensive function. Let p represent the dimension of the function space currently under consideration. As shown in step 1 of Fig. 2, it is initialized to M. In step 2 of Fig, DEC-BASIC is invoked with A and G to obtain A^* and L as described. As shown in Fig., the algorithm terminates if A^*L = G or if further aggregation of functions is not possible. Otherwise, the dimension of the function space is reduced by 1 as shown in steps 3 and 4 of Fig. In step 3, the p^{th} row of G is added to the (p-1)th row of G and the p^{th} row is then set to zero. This represents the aggregation of the two least resource intensive functions in the current set of functions. Step 4 shows the same aggregation operation performed on A. Step 5 updates p to reflect the reduction in the dimension of the function space. This is followed by a re invoking of DEC-BASIC with the new values of A and G. DEC-ITERATIVE has a maximum number of M- 1 iterations. The first iteration of the method is essentially the same as DEC-BASIC, which includes all the functions offered by the system. At the nth iteration, the n least resource intensive functions are aggregated as one function. Note that, as with DEC-BASIC, DEC-ITERATIVE is not guaranteed to provide an exact solution. If by the end of the M - 1 iteration, the method was not able to provide a solution while satisfying the constraints, the L computed in the last iteration is reported as the final solution from DECITERATIVE.

DECITERATIVE can improve on the predictive accuracy of DEC-BASIC for cases where the latter technique could not achieve an exact match of the desired workload ix. In particular, aggregating functions with low resource demands allows the algorithm to discover solutions where the more resource intensive functions are matched better thereby resulting in improved accuracy. It must be noted that DEC-ITERATIVE differs from other techniques, which employ function aggregation. In effect, DEC-ITERATIVE attempts to first find the best possible match, but incrementally relaxes the degree of match when an exact match is not possible. To the best of our knowledge, other techniques do not support such a systematic and automated technique to aggregate functions.

DEC-ITERATIVE ALGORITHM

1. Initialize dimension of function space p=M
2. Invoke DEC-BASIC in which [A*, L]= INVOKE_DEC_BASIC(A,G)
3. Check whether A*L = = G or p=2.
4. If the above condition is satisfied discard the process.
5. Else aggregate 2 least resource intensive functions in G by initialising $G_{p,1} = 0$ and $G_{p-1,1} = G_{p-1,1} + G_{p,1}$
6. Aggregate 2 least resource intensive functions in A by initialising $A_{i,p} = 0$ and $A_{i,p-1} = A_{i,p-1} + A_{i,p}$
7. After completing the process reduce dimension of function space with p=p-1
8. Then Goto step:2

III. THE FRAMEWORK

A. Demand Parameters Estimation

Asymptotic normal approximations was developed to the impact of demand parameter uncertainty on the service level variance, and discuss the distributional characteristics of the product demands, the critical fractile, and the length of the historical demand data for which it is critical to capture demand parameter uncertainty. This section describes two variants of the DEC technique. DEC-BASIC variant that determines a linear combination of benchmarks to match a desired mix of system functions. An extension called DEC-ITERATIVE that handles in an automated manner scenarios where an exact match of a desired mix is not possible with the available set of benchmarks. A way of capturing the demand parameter uncertainty in the inventory simulation is to use a Bayesian model that samples the demand parameters from their Bayesian posterior density functions before each simulation replication. There exists well-established literature on Bayesian probability theory for representing the uncertainty around the parameters of the standard families of distributions (Gelman et al. 2000). Assuming the availability of limited historical demand data, this literature is used to obtain an asymptotic variance approximation for the demand parameter uncertainty. For the demand random variable X having the normal distribution with mean μ and variance σ^2, the likelihood function of the available historical demand data x_t, t=1, 2, . . . ,n is given by

$$f(x_1, x_2, \dots, x_n \mid \mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left\{-\frac{1}{2\sigma^2}\sum_{t=1}^{n}(x_t - \mu)^2\right\} \quad (3)$$

Using a normal prior density function with mean μ_0 and variance σ^2/κ_0 for the mean μ conditional on the variance σ^2, i.e,

$$\mu \mid \sigma^2 \propto \frac{1}{(\sigma^2)^{1/2}} \exp\left\{-\frac{K_0}{2\sigma^2}(x_t - \mu)^2\right\} \quad (4)$$

and an inverse gamma prior density function with shape parameter $\nu_0/2$ and scale parameter $\varsigma_0^2/2$ for the variance σ^2, i.e.,

$$\sigma^2 \propto \frac{1}{(\sigma^2)^{(v_0+2)/2}} \exp\left\{-\frac{\varsigma_0^2}{2\sigma^2}\right\} \quad (5)$$

Obtain a joint conjugate prior density function, $\pi(\mu,\sigma^2)$ for the demand parameters μ and σ^2:

$$\pi(\mu, \sigma^2) \propto \frac{1}{(\sigma^2)^{\frac{v_0+3}{2}}} \exp\left\{-\frac{1}{2\sigma^2}(\varsigma_0^2 + (\mu - \mu_0)^2\right\} \quad (6)$$

B. The Historical Data Repository (HDR)

The HDR is a flexible, generic component implemented by SZTAKI to provide facilities for log data collection and on-demand predictions. The HDR is implemented in Java and can be embedded into Java code or can be run as a separate Web Service. In both cases, other software components can send their log data to the HDR, either by our customizable client APIs or via direct calls. Incoming information must be in the format of RDF [9], based on the core ontologies available to all components. The conversion to RDF can be implemented in client APIs if necessary. Thus, in our scenario, the service collects and stores information about past events (i.e., job executions and resource usage), and provides mining and searching for these mentioned past events. The collected status information is stored as RDF inside the HDR. The repository can then be used for extracting statistics- and knowledge-based information or predictions.

For example, a predictor can build a classification model on top of the results of a semantic query. The classifier can then be used to provide predictions based on the past. In our specific scenario, the HDR is loaded with the description of the executed jobs, their resource usage and the resource downtimes. This data is used to train the classifier in order to provide estimations on the probabilities of delays in the schedule and problems with the job completion and estimations on the reliability of the used resources. As the statistical model is periodically updated, the provided values dynamically reflect the experiences of the recently finished executions.

Benchmarking is the process of comparing one's demand estimation processes and performance metrics to industry bests or best practices from other industries. Dimensions typically measured are quality, time and cost. In the process of benchmarking, management identifies the best firms in their estimated demand, or in another estimation where similar processes exist, and compare the results and processes of those studied (the "targets") to one's own results and processes. In this way, here learn how well the targets perform and, more importantly, the business processes that explain why these firms are successful.

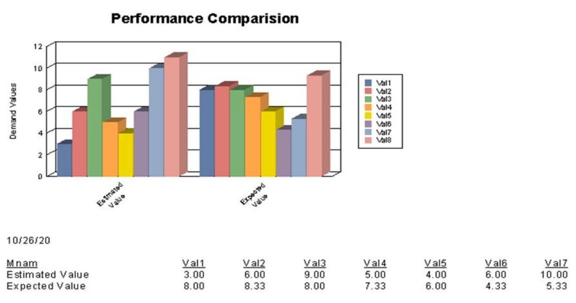
C. Semantic Resource Allocation Process

In this case, we require ontology for modeling the system entities such as customers, service providers, jobs and resources and other important data such as resource allocation data for job scheduling and historical data for making predictions. SVM analyze data in ontology and used for classification. i.e. the SVM makes the possible classes from the requirement. From the possible classes, the resources were allocated to the application.

IV. PERFORMANCE EVALUATION

To evaluate the efficiency, we measure the CPU time of the process.

A. PERFORMANCE COMPARISION



10/26/20

Mnam	Val1	Val2	Val3	Val4	Val5	Val6	Val7
Estimated Value	3.00	6.00	9.00	5.00	4.00	6.00	10.00
Expected Value	8.00	8.33	8.00	7.33	6.00	4.33	5.33

Figure 1: Performance Comparison

B. DEMAND VALUES

Demand Values shown below are estimated and expected demand value's based on the CPU time.

TABLE 1

DEMAND VALUES

Functions	Estimated Value	Expected Value
20	3	8
40	6	8.3333
60	9	8
35	5	7.33333
30	4	6
46	6	4.33333
70	10	5.33333
78	11	9.333333

V. CONCLUSIONS

Demand Estimation with Confidence (DEC) is a technique for estimating the resource demands of services that are implemented by multitier systems. A customer workload mix was expressed as a linear combination of a subset of existing benchmarks. This technique predicted the aggregate resource demands of new workload mixes directly. Linear combination weights for the benchmarks were used along with the measured demands of those benchmarks for offering a demand estimate for the specified mix.

We present a generic approach and a re-usable solution for the collection and exploitation of historical log data produced by services. The heterogeneity of log data arriving from various resources calls for a semantic data representation, which can facilitate the unification of these data and a query mechanism supported by inference. Our approach demonstrates the coupling of semantic data processing with data mining as a promising novel combination.

REFERENCES

[1] Amazon Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2/, 2011.
[2] C. Amza, A. Chanda, A.L. Cox, S. Elnikety, R. Gil, K. Rajamani, W.Zwaenepoel, E. Cecchet, and J. Marguerite, "Specification and Implementation of Dynamic Web Site Benchmarks," Proc. Fifth IEEE Workshop Workload Characterization, pp. 3-13, Nov. 2002.
[3] Y. Bard and M. Shatzoff, "Statistical Methods in Computer Performance Analysis," Current Trends in Programming Methodology, vol. 3, pp. 1-51, Prentice-Hall, 1978.
[4] G. Casale, E.Z. Zhang, and E. Smirni, "Kpc-Toolbox: Simple Yet Effective Trace Fitting Using Markovian Arrival Processes," Proc. Fifth Conf. Quantitative Evaluation of Systems, pp. 183-187, Sept. 2008.
[5] Y. Dodge and J. Jureckova, Adaptive Regression. Springer2000.
[6] N.R. Draper and H. Smith, Applied Regression Analysis. John Wiley & Sons, 1998.
[7] J.J. Dujmovic, "Universal Benchmark Suites," Proc. Seventh Int'l Symp. Modeling, Analysis and Simulation of Computer and Telecomm. Systems, pp. 197-205, 1999.
[8] Eng. Statistics Handbook, http://www.itl.nist.gov/div898/ handbook/, 2011.
[9] R. Jain, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. John Wiley & Sons, 1991.
[10] S. Kraft, S. Pacheco-Sanchez, G. Casale, and S. Dawson, "Estimating Service Resource Consumption from Response Time Measurements," Proc. Fourth Int'l Conf. Performance Evaluation Methodologies and Tools, Oct. 2009.
[11] D. Krishnamurthy, "Synthetic Workload Generation for Stress Testing Session-Based Systems," PhD thesis, Dept. of Systems and Computer Eng., Carleton Univ., 2004.
[12] D. Krishnamurthy, J.A. Rolia, and S. Majumdar, "A Synthetic Workload Generation Technique for Stress Testing Session-Based Systems," IEEE Trans. Software Eng., vol. 32, no. 11, pp. 868-882, Nov. 2006.
[13] U. Krishnaswamy and D. Scherson, "A Framework for Computer Performance Evaluation Using Benchmark Sets," IEEE Trans. Computers, vol. 49, no. 12, pp. 1325-1338, Dec. 2000.
[14] T. Kubokawa and M. Srivastava, "Improved Empirical Bayes Ridge Regression Estimators under Multicollinearity," Comm. In Statistics—Theory and Methods, vol. 33, no. 8, pp. 1943-1973, Dec. 2004.
[15] Y. Lu, T. Abdelzaher, C. Lu, L. Sha, and X. Liu, "Feedback Control with Queuing-Theoretic Prediction for Relative Delay Guarantees in Web Servers," Proc. IEEE Real-Time and Embedded Technology and Applications Symp., pp. 208-217, 2003.
[16] D. Menasce, "Computing Missing Service Demand Parameters for Performance Models," Proc. Int'l Computer Measurement Group Conf., pp. 241-248, 2008.
[17] N. Mi, Q. Zhang, A. Riska, E. Smirni, and E. Riedel, "Performance Impacts of Autocorrelated Flows in Multi-Tiered Systems," Performance Evaluation, vol. 64, nos. 9-12, pp. 1082-1101, 2007.
[18] D. Mosberger and T. Jin, "httperf—A Tool for Measuring Web Server Performance," ACM SIGMETRICS Performance Evaluation Rev., vol. 26, no. 3, pp. 31-37, 1998.