DYNAMIC SEMI-MARKOVIAN WORKLOAD MODELING

Nima Sharifimehr and Samira Sadaoui
Department of Computer Science, University of Regina, Regina, Canada

Keywords: Markovian Models, Workload Modeling, Enterprise Application Servers.

Abstract: In this paper, we present a Markovian modeling approach which is based on a combination of existing Semi-Markov and Dynamic Markov models. The proposed approach is designed to be an efficient statistical modeling tool to capture both actions intervals patterns and sequential behavioral patterns. A formal definition of this model and detailed algorithms for its implementation are illustrated. We show the applicability of our approach to model workload of enterprise application servers. However, the given formal definition of our proposed approach prepares a firm ground for academic researchers to investigate many other possible applications. Finally, we prove the accuracy of our dynamic semi-Markovian approach for the most chaotic situations.

1 INTRODUCTION

The performance of software systems is significantly affected by the incoming workload toward them (Rolia et al., 2006). To improve the performance of such systems, we need to analyze the workload they process. Through workload modeling, we can find the bottlenecks and issues which reduce the throughput of software systems. However, analyzing real workload is an extremely difficult task and on the other hand reproducing workload with specific features is not easily feasible (R.J. Honicky and Sawyer, 2005). Using modeling approaches to build a generic workload model for a given system is an appropriate solution to control the complexity of workload analysis. Markovian models have been widely used for the purpose of modeling workload processed by software systems (Dhyani et al., 2003) (Eirinaki et al., 2005) (Sarukkai, 2000). Also some combinatory approaches have been investigated, which take advantage of combining Markovian models with other techniques such as clustering (F. Khalil and Wang, 2007), and neural networks (Firoiu and Cohen, 2002).

In this paper we are interested in workload modeling for enterprise application servers. Enterprise application servers as server programs in distributed infrastructures provide development and deployment facilities to integrate all organizations' applications and back-end systems (Mariucci, 2000). We view the workload modeling as the process of analyzing the sequence of incoming requests and finding patterns across method invocations on hosted applications and components. Investigated approaches in the literature on workload modeling only focus on extraction of sequential patterns of incoming requests. They do not discuss solutions for capturing patterns of intervals between incoming requests and their process times. Therefore, in this paper we illustrate a Dynamic Semi-Markov Model (*DSMM*) which can be used to efficiently model the incoming workload toward an enterprise application server. Our modeling approach enhances current Markov modeling solutions (R.J. Honicky and Sawyer, 2005) (Song et al., 2004) for accurate workload modeling. *DSMM* captures patterns of intervals between incoming requests, the required times to process them, and also their sequential model. In other words, it finds patterns across incoming requests and also statistical models of requests intervals and process times. Measuring the accuracy of our proposed approach for the most chaotic situations, we show that DSMM-based workload modeling for enterprise application servers is a very accurate solution.

The rest of this paper is structured as follows. Section 2 introduces our dynamic-semi Markov Model. Section 3 explains the dynamic building process to build and adjust a *DSMM*. Section 4 illustrates the results of experimentations that evaluate the accuracy of

Sharifimehr N. and Sadaoui S. (2008).
DYNAMIC SEMI-MARKOVIAN WORKLOAD MODELING.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - DISI*, pages 125-130
DOI: 10.5220/0001678801250130

*DSMM*s. Finally, Section 5 concludes the paper with some future directions.

2 DYNAMIC SEMI-MARKOV MODEL

This section introduces a Dynamic Semi-Markov Model *(DSMM)* which dynamically adapts itself to the analyzing data. *DSMM* is designed with this goal in mind to be an accurate and efficient workload modeling tool for enterprise application servers. A *DSMM* is a Kripkle or temporal structure (van der Hoek et al., 2005) in which transitions and states are labeled with probability distributions. The probability distributions attached to a transition describe the speed of evolution from one state to another and also the probability of the occurrence of that transition in terms of discrete distribution functions (Zellner, 2004). The probability distribution attached to a state is also a discrete distribution function but it describes the idle time spent when a transition evolves the system to a new state. Consequently, being a change in the system - a transition from the current state to another - depends on the attached time distribution functions to states and transitions. This dependency which is in contradiction with Markov property (Cover and Thomas, 2006) makes *DSMM* a semi-Markov (López et al., 2001) rather than Markov. In the following comes the formal definition of *DSMM*.

Definition 1. *(Dynamic Semi-Markov Model)*

Let AP be a finite set of tokens seen in the analyzing data. A *DSMM* is a 5-tuple $(S,\ T,\ P,\ PDF_S,\ PDF_T)$ where:

- S is a finite set of states.
- T is a finite set of discrete time units $[t_{min}, t_{max}]$.
- P: $S \times AP \rightarrow S \times [0,1]$ is the transition function which states the probability of evolving the system from a state to another when a specific element is seen in the analyzing data.
- PDF_S: $S \times T \rightarrow [0,1]$ is the idle time probability matrix (satisfying $\sum_{t \in T} PDF_S(s,t) = 1$ for each $s \in S$).
- PDF_T: $S \times AP \times T \rightarrow [0,1]$ is the probability matrix (satisfying $\sum_{t \in T} PDF_T(s,a,t) = 1$ for each $s \in S$ and $a \in AP$) for the evolution speed of the system when a specific element is seen in the analyzing data.

Figure 1 shows an example of *DSMM* which only contains two states of S_0 and S_1 and the tokens of the analyzing data belongs to the finite set of $\{0,1\}$.

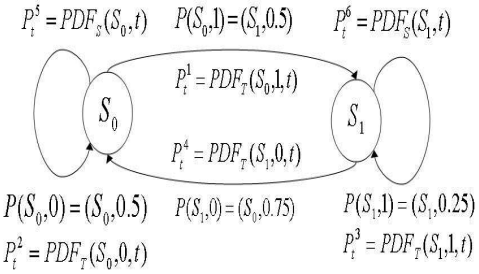

Figure 1: An example of *DSMM*.

3 DYNAMIC PROCESS OF BUILDING A DSMM

The formal definition of *DSMM* mentioned above describes the static structure of a *DSMM* and the example shown afterward illustrates how a typical *DSMM* looks like at a time snapshot of its lifecycle. However, one significant aspect of a *DSMM* is its dynamic nature. A *DSMM* changes dynamically during its lifecycle to find the best fit model for a specific system. Dynamic process of building a *DSMM* brings two significant advantages:

- When all or some internal states of a system are unknown, this dynamic building process starts building the *DSMM* with only one state and then extracts the others through the statistical analysis of the processing data. This speeds up the modeling process and makes it possible to easily model systems with unperceivable internal states.
- A *DSMM* is useful even before being close enough to the best fit model. In other words, a *DSMM* can be used during the dynamic building process when it is not as accurate as final built model. This feature is so critical for real-time systems where late results of a process are considered incorrect (Shaw, 2000).

Before giving a formal definition for the best fit model, an accuracy measurement tool is required. For this sake, it is shown how to measure Root Mean Square Error (RMSE) (Zellner, 2004) of a *DSMM* based on a stream of sample data taken from the modeling system.

3.1 Rmse of a DSMM

Let assume a stream of analyzing data from a specific system is described as a finite sequence of 3-tuples $(I_0,A_0,E_0) \rightarrow ... \rightarrow (I_i,A_i,E_i) \rightarrow ... \rightarrow (I_n,A_n,E_n)$ where I_i shows the idle time before the observation of the next token in the stream of analyzing data, A_i is the

next token which system emits after spending the idle time, and E_i is the evolution speed which describes the time that system spends to emit A_i. Processing the tuples of this sequence through the steps mentioned below generates a prediction sequence which will be used directly to compute the *RMSE* of a model:

Algorithm 1. *Prediction Sequence Generation*

1. Define S as a state
2. $S \leftarrow$ Starting state of *DSMM*
3. $j \leftarrow 0$
4. For each $i \in [0,n]$
   (a) Define P_t^j, P_t^{j+1}, P_t^{j+2} as probabilities
   (b) $P_t^j \leftarrow PDF_S(S,I_i)$
   (c) $P_t^{j+1} \leftarrow P(S,A_i)$
   (d) $P_t^{j+2} \leftarrow PDF_T(S,A_i,E_i)$
   (e) Add P_t^j, P_t^{j+1}, and P_t^{j+2} to the prediction sequence
   (f) $S \leftarrow P(S,A_i)$
   (g) $j \leftarrow j+3$

Subsequently using the resulted prediction sequence generated through Algorithm 1, *RMSE* of a model can be computed through the following formula (Zellner, 2004):

$$RMSE = \sqrt{\frac{\sum_{i=0}^{n*3}(P_t^i - 1)^2}{n*3}}$$

Following comes the formal definition of the best fit model which defines the ideal goal of building a *DSMM*.

Definition 2. *(The Best Fit Model)*

A *DSMM* is the best fit model of a specific system when its *RMSE* for any sample data from that system equals to zero.

However, building the best fit models in the real world is almost impossible in most cases. Therefore, the ideal goal of modeling a system can be stated as building the closest model to the best fit model. In other words, the modeling process aims to reduce the *RMSE* of a model as much as possible.

3.2 Building a DSMM

Having the formal definition of the ideal goal for modeling a system, it is time to dig into the dynamic process of building a model. Algorithm 2 used for this purpose is based on the dynamic modeling approach used in Dynamic Markov Compression (DMC) (Ormack and Horspool, 1987). Also the applicability and efficiency of this modeling approach has been proven before on workload modeling with the purpose of developing a predictive automatic tuning service for object pools (Sharifimehr and Sadaoui, 2007). Though it is heavily modified to be applicable for the semi-Markov model proposed above. Basically the algorithm starts with a *DSMM* which only contains one state S_0 and this starting *DSMM* can be illustrated as follows (let assume $AP = \{a,b\}$) :

- $P_t^1 = PDF_S(S_0,t)$ where $PDF_S(S_0,t)$ is a uniform discrete distribution, so we can say for all values of $t \in T$ we have $PDF_S(S_0,t) = 1/(t_{max} - t_{min})$ which also satisfies $\sum_{t=t_{min}}^{t_{max}} PDF_S(S_0,t) = 1$.
- $P(S_0,a) = P(S_0,b) = (S_0,0.5)$ which states that the observation of all allowed tokens (i.e. a and b) in the analyzing data evolves the system back to S_0 and the probability of observation of each is equal.
- $P_t^2 = PDF_T(S_0,a,t)$ and $P_t^3 = PDF_T(S_0,b,t)$ where both of them are again uniform discrete distributions, so we have $PDF_T(S_0,a,t) = PDF_T(S_0,b,t) = 1/(t_{max} - t_{min})$ and also $\sum_{t=t_{min}}^{t_{max}} PDF_T(S_0,a,t) = \sum_{t=t_{min}}^{t_{max}} PDF_T(S_0,b,t) = 1$.

Subsequently we set the only state of *DSMM* mentioned above as the current state, namely S_C. Then for each 3-tuple i.e. (I_i,A_i,E_i) in the analyzing data, we go through the steps of Algorithm 2 , mentioned below. Algorithm 2 dynamically evolves the *DSMM* into a model closer to the best fit model.

Algorithm 2. *Building a DSMM Dynamically*

1. Update the idle time distribution of the current state based on the idle time I_i. To store idle time distributions, each *DSMM* uses a two dimensional array: $IDLE[S_0..S_{MAX}][t_{min}..t_{max}]$. Consequently, to figure out the probability of spending t_i time units in state S_C, the following formula can be used:

   $$PDF_S(S_C,t_i) = IDLE[S_C][t_i] / \sum_{t=t_{min}}^{t_{max}} IDLE[S_C][t]$$

   So, to update the idle time distribution of state S_C when the idle time I_i is in the current processing 3-tuple of the analyzing data, we need to only perform the following operation:

   $$IDLE[S_C][t_i] \leftarrow IDLE[S_C][t_i] + 1$$

2. Update the transition time distribution of the transition which evolves the system to its next new state based on the observation of A_i which takes E_i time duration. A three dimensional array $SPEED[S_0..S_{MAX}][A_0..A_{|AP|}][t_{min}..t_{max}]$ is used to store transition time distributions for each *DSMM*. Therefore, to compute the probability of evolving

from state S_C after the observation of A_i within E_i time, we have:

$$PDF_T(S_C, A_i, E_i) = SPEED[S_C][A_i][E_i] / \sum_{t=t_{min}}^{t_{max}} SPEED[S_C][A_i][t]$$

So, to update the transition time distribution of evolution from the current state S_C with observation of A_i when the transition time E_i is in the current processing 3-tuple of the analyzing data, we need to only perform the following operation:

$$SPEED[S_C][A_i][E_i] \leftarrow SPEED[S_C][A_i][E_i] + 1$$

3. Update the transition probability of evolving from the current state as a result of the observation of A_i. For this sake, there is a couple of two dimensional arrays: $MOVE[S_0..S_{MAX}][A_0..A_{|AP|}]$ and $COUNT[S_0..S_{MAX}][A_0..A_{|AP|}]$ attached to each *DSMM*. The array $MOVE$ shows which state the system evolves to from each state after the observation of each token and the array $COUNT$ keeps track of the number of times each token has been observed in each state of the model. These two arrays are the building elements of the transition function P:

$$P(S_C, A_i) = (MOVE[S_C][A_i], COUNT[S_C][A_i] / \sum_{a \in AP} COUNT[S_C][a])$$

Then updating the transition probability of evolving from the current state after the observation of A_i is done by:

$$COUNT[S_C][A_i] \leftarrow COUNT[S_C][A_i] + 1$$

4. If it is suitable add a new state(s) to the model to reduce its *RMSE* and afterwards adjust involved parameters of the model. Adding a new state to the model which we refer to as cloning is an effective way to improve its fitness. The idea is when transition to the state $S_N = MOVE[S_C][A_i]$ from the current state S_C is more probable than any other state, we clone the state S_N to capture more details about that specific evolution in the system. We can use the cloning conditions proposed in DMC (Ormack and Horspool, 1987) and adapt it to the definition of *DSMM*. So when the following conditions are satisfied cloning happens:

   (a) $Threshold_1 \leq COUNT[S_C][A_i]$

   (b) $Threshold_2 \leq \sum_{a \in AP} COUNT[S_N][a] - \sum_{a \in AP} COUNT[S_C][a]$

   First condition is to make sure the transition from S_C to S_N has taken enough times (i.e. more than $Threshold_1$ times). And the second condition is to assure that transitions to S_N from S_C is more probable than any other state and $Threshold_2$ is used to describe this condition.

5. Change the current state of the model according to the observed token A_i. Having the array $MOVE$ as explained above based on the observation A_i, this change of state is carried out as follows:

$$S_C \leftarrow MOVE[S_C][A_i]$$

3.3 Adjustment of Parameters

To model real-world systems accurately and efficiently using *DSMMs*, we need to take the following points into account when adjusting parameters of Algorithm 2 proposed above (Ormack and Horspool, 1987):

- Choosing small values for $Threshold_1$ and $Threshold_2$ will increase the growth speed of the model. In contrast, assigning large values to these parameters decrease the speed of adding new states to the model. Adding more states to the model helps to capture more details about the behavior of the system. However, it does not mean necessarily that the model will get closer to the best fit model faster. Unfortunately there are no absolute values for these parameters which can be used to model all systems. An applicable way is to find their values specifically for each system through a feed-back based approach. A typical feed-back based solution measures the accuracy of the model actively and whenever the accuracy is not showing any change, the values of $Threshold_1$ and $Threshold_2$ will be reduced.
- Adjusting the maximum number of states, namely MAX_S, depends on available memory accessible to the modeling process. However, an issue which should be addressed is how to react when the number of states in a model reaches this maximum. Dynamic Markov Compression (DMC) (Ormack and Horspool, 1987) suggests to discard the model and start the modeling process from the scratch again. Though this approach is useful when the modeled behavior is not going to be used in the future. Otherwise, we can save the current model into a persistent storage and then start with a new model.

4 EVALUATION RESULTS

We conduct our experimentations using synthetic workload models defined by RUBiS (Cecchet et al., 2002) to evaluate the accuracy of the proposed approach for workload modeling using *DSMMs*. These workload models are designed according to standards introduced in TPC-W (Garcia and Garcia, 2003)

which model an online bookstore (see clauses 5.3.1.1. and 6.2.1.2 of the TPC-W v1.6 specification (Council, 2001)). For this purpose, we use JOnAS J2EE (Sicard et al., 2006) as our enterprise application server and integrate an incoming request monitor into it as is shown in Figure 2. We use a machine equipped with PIV 2.80 GHz CPU, 1 GB of RAM. Also we use MSSQL Server 2000 as the backend DBMS running on MS Windows Server 2003 Service Pack 1. In this architecture, we use RUBiS client module to simulate our workload generators. The RUBiS client module generates workloads with high randomness which simulates the most chaotic situation. Therefore, the results of our experimentations show the accuracy of *DSMM*s for the worst case scenarios.

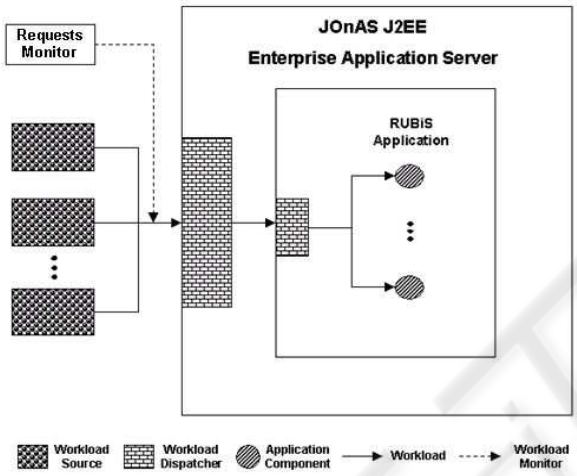

Figure 2: Integration of an incoming requests monitor into JOnAS J2EE which hosts RUBiS application.

Algorithm 2 is only capable of processing a sequence of 3-tuples $(I_0,A_0,E_0) \rightarrow .. \rightarrow (I_i,A_i,E_i) \rightarrow ... \rightarrow (I_n,A_n,E_n)$. Therefore, we implement a converter module which is responsible for generating this sequence from a monitored workload.

To evaluate the accuracy of our approach which is based on *DSMM*, we use sample data tracing. For this purpose, at first we build a *DSMM* for workload generated by RUBiS clients according to a specific workload model. Afterward, we generate another sequence of workload based on the same workload model and trace it on the built *DSMM*. The tracing process will generate a sequence of 3-tuples $(P_I^0,P_A^0,P_E^0) \rightarrow ... \rightarrow (P_I^i,P_A^i,P_E^i) \rightarrow ... \rightarrow (P_I^n,P_A^n,P_E^n)$. Then the average difference between the *DSMM* and real workload model can be measured as follows:

$$difference = \frac{\sum_{i=0}^{n} \frac{(1-P_I^i)+(1-P_A^i)+(1-P_E^i)}{3}}{n}$$

And as the similarity between the *DSMM* and the real workload equals to the complement of the average difference, we have:

$$similarity = 1 - difference$$

Table 1: Similarity of built DSMMs and real workloads.

Workload	MAX_S = 10	MAX_S = 50	MAX_S = 100	MAX_S = 500
W0	0.36	0.45	0.45	0.49
W1	0.37	0.41	0.45	0.49
W2	0.37	0.44	0.45	0.49
W3	0.34	0.43	0.46	0.50
W4	0.36	0.44	0.46	0.49
W5	0.34	0.42	0.47	0.48
W6	0.36	0.41	0.45	0.49
W7	0.37	0.43	0.46	0.49
W8	0.37	0.42	0.42	0.49
W9	0.35	0.45	0.45	0.49

Table. 1 shows the resulted similarity of built *DSMM*s for each generated workload i.e. W_i. It also illustrates the relationship between size of *DSMM*s and their measured similarity for different workloads.

Interestingly, increasing the size of a *DSMM* increases the similarity only for a while. In other words, the accuracy of a *DSMM* becomes stable after a specific size and does not increase anymore. These results show that DSMM-based workload modeling can reach the accuracy of approximately 50% in the most chaotic situation which the incoming workload has high randomness. Therefore, our proposed approach based on *DSMM* is an efficient and accurate workload modeling solution for enterprise application servers.

5 CONCLUSIONS

In this paper, we have introduced a dynamic semi-Markovian approach to model the incoming workload toward an Enterprise Application Server. *DSMM* is designed to be an efficient workload modeling tool for enterprise application servers. A formal definition of the *DSMM* independent of its proposed application in this paper is given. This formal definition allows other researchers to investigate other possible applications of *DSMM*.

Afterwards, step by step procedure of building a *DSMM* for a typical system is illustrated. The brief explanation of this procedure assures that sufficient information is provided for real-world applications. Also to make the *DSMM* useful for a wide range of applications, adjustment techniques has been discussed.

Finally, to prove the accuracy of the proposed approach for the real-world applications, experiments

for the most chaotic situations are carried out. The result of experiments show that DSMM-based workload modeling for enterprise application servers is a very accurate solution.

REFERENCES

Cecchet, E., Marguerite, J., and Zwaenepoel, W. (2002). Performance and scalability of ejb applications. In *OOPSLA '02: Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 246–261, New York, NY, USA. ACM Press.

Council, T. P. P. (2001). *TPC Benchmark W, Standard Specification*.

Cover, T. M. and Thomas, J. A. (2006). *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience.

Dhyani, D., Bhowmick, S. S., and Ng, W.-K. (2003). Modelling and predicting web page accesses using markov processes. In *DEXA '03: Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, page 332, Washington, DC, USA. IEEE Computer Society.

Eirinaki, M., Vazirgiannis, M., and Kapogiannis, D. (2005). Web path recommendations based on page ranking and markov models. In *WIDM '05: Proceedings of the 7th annual ACM international workshop on Web information and data management*, pages 2–9, New York, NY, USA. ACM.

F. Khalil, J. L. and Wang, H. (2007). Integrating markov model with clustering for predicting web page accesses. In *AusWeb'07: Proceedings of the 13th Australian World Wide Web conference*, Australia.

Firoiu, L. and Cohen, P. R. (2002). Segmenting time series with a hybrid neural networks - hidden markov model. In *Eighteenth national conference on Artificial intelligence*, pages 247–252, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Garcia, D. F. and Garcia, J. (2003). Tpc-w e-commerce benchmark evaluation. *Computer*, 36(2):42–48.

López, G. G. I., Hermanns, H., and Katoen, J.-P. (2001). Beyond memoryless distributions: Model checking semi-markov chains. In *PAPM-PROBMIV '01: Proceedings of the Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification*, pages 57–70, London, UK. Springer-Verlag.

Mariucci, M. (2000). Enterprise application server development environments. Technical report, Institute of Parallel and Distributed High Performance Systems.

Mosbah, M. and Saheb, N. (1997). A syntactic approach to random walks on graphs. In *WG '97: Proceedings of the 23rd International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 258–272, London, UK. Springer-Verlag.

Ormack, G. V. and Horspool, R. N. S. (1987). Data compression using dynamic markov modelling. *Comput. J.*, 30(6):541–550.

R.J. Honicky, S. R. and Sawyer, D. (2005). Workload modeling of stateful protocols using hmms. In *proceedings of the 31st annual International Conference for the Resource Management and Performance Evaluation of Enterprise Computing Systems*, Orlando, Florida.

Rolia, J., Cherkasova, L., and Friedrich, R. (2006). Performance engineering for enterprise software systems in next generation data centres. Technical report, HP Laboratories.

Sarukkai, R. R. (2000). Link prediction and path analysis using markov chains. *Comput. Networks*, 33(1-6):377–386.

Sharifimehr, N. and Sadaoui, S. (2007). A predictive automatic tuning service for object pooling based on dynamic markov modeling. In *Proc. of 2nd International Conference on Software and Data Technologies*, Barcelona, Spain.

Shaw, A. C. (2000). *Real-Time Systems and Software*. John Wiley & Sons, Inc., New York, NY, USA.

Sicard, S., Palma, N. D., and Hagimont, D. (2006). J2ee server scalability through ejb replication. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 778–785, New York, NY, USA. ACM.

Song, B., Ernemann, C., and Yahyapour, R. (2004). Parallel computer workload modeling with markov chains. In *Job Scheduling Strategies for Parallel Processing*, pages 47–62, London, UK. Springer Verlag LNCS.

van der Hoek, W., Jamroga, W., and Wooldridge, M. (2005). A logic for strategic reasoning. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 157–164, New York, NY, USA. ACM Press.

Vose, M. D. (1991). A linear algorithm for generating random numbers with a given distribution. *IEEE Trans. Softw. Eng.*, 17(9):972–975.

Zellner, A. (2004). *Statistics, Econometrics and Forecasting*. Cambridge University Press.