Presentation for CS 895 Fall ’06 on:

Resource Allocation for Autonomic Data Centers Using Analytic Performance Models

Proceedings of the Second International Conference on Autonomic Computing (ICAC’05)
Resource Allocation for Autonomic Data Centers Using Analytic Performance Models
Bennani, M.N.; Menasce, D.A. Page(s): 229- 240

Copyright

- The content within this presentation is based on the contents of the ICAC ’05 paper entitled: Resource Allocation for Autonomic Data Centers using Analytic Performance Models Bennani, M.N.; Menasce, D.A. Page(s): 229- 240
- This presentation incorporates figures and content licensed by individuals, companies, or organizations that are protected by U.S. and foreign copyright laws. All persons reproducing, redistributing, or making commercial use of this information are expected to adhere to the terms and conditions asserted by the copyright holder.
- Transmission or reproduction of protected items beyond that allowed by fair use (PDF) as defined in the copyright laws requires the written permission of the copyright owners

Contents

- Intro
- Dynamic Resource Allocation Problem
- Controller Approach
- Performance Model for Online Transaction AEs
- Performance model for Batch Processing AEs
- The Experimental Setting
- Results
- Conclusion

Introduction

- *Large data centers host several application environments (AEs) that are subject to workloads of varying and unpredictable intensities.*
- *Data center servers may need to be dynamically redeployed among the various AEs to optimize global utility.*
- *Previous approaches :*
  - *suffer from scalability limitations*
  - *and do not address the possibility of having multiple classes of workloads executing within an AE.*
- *These limitations are addressed using analytic queuing network models combined with combinatorial beam search techniques.*
- *The effectiveness of this approach is demonstrated via simulation experiments.*
- *Both online and batch workloads are considered*

Dynamic Resource Allocation Problem: Application Environments

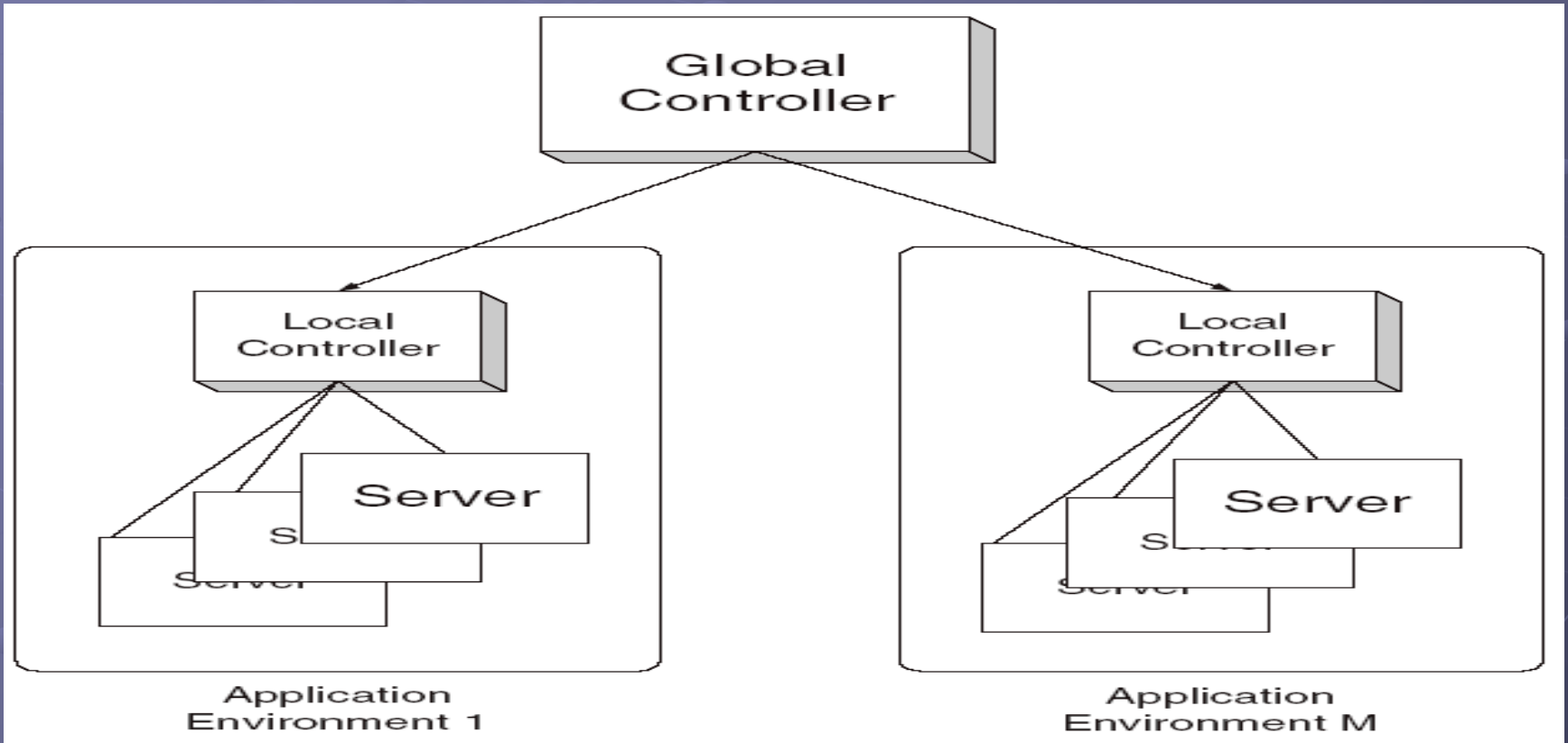


Figure 1. Application Environments.

Formulas

$$\vec{w}_i = \begin{cases} (\lambda_{i,1}, \cdots, \lambda_{i,S_i}) & \text{if } AE_i \text{ is online} \\ (c_{i,1}, \cdots, c_{i,S_i}) & \text{if } AE_i \text{ is batch} \end{cases} \quad (1)$$

$\vec{R}_i = (R_{i,1}, \cdots, R_{i,S_i})$ and $\vec{X}_i = (X_{i,1}, \cdots, X_{i,S_i})$.

$$\vec{R}_i = \mathcal{M}_i^o(\vec{w}_i, n_i) \quad (2)$$

$$\vec{X}_i = \mathcal{M}_i^b(\vec{w}_i, n_i) \quad (3)$$

$$U_i = \begin{cases} f(\vec{R}_i) = f(\mathcal{M}_i^o(\vec{w}_i, n_i)) & \text{if } AE_i \text{ is online} \\ g(\vec{X}_i) = g(\mathcal{M}_i^b(\vec{w}_i, n_i)) & \text{if } AE_i \text{ is batch.} \end{cases}$$

$$U_g = h(U_1, \cdots, U_M). \quad (5)$$

$$U_{i,s} = \frac{K_{i,s}.e^{-R_{i,s}+\beta_{i,s}}}{1 + e^{-R_{i,s}+\beta_{i,s}}} \quad (6)$$

$$U_i = \sum_{s=1}^{S_i} a_{i,s} \times U_{i,s} \quad (7)$$

where $0 < a_{i,s} < 1$ and $\sum_{s=1}^{S_i} a_{i,s} = 1$

$$U_{i,s} = K_{i,s} \times \left(\frac{1}{1 + e^{-X_{i,s}+\beta_{i,s}}} - \frac{1}{1 + e^{\beta_{i,s}}} \right) \quad (8)$$

Fig 2: utility as a function of response time

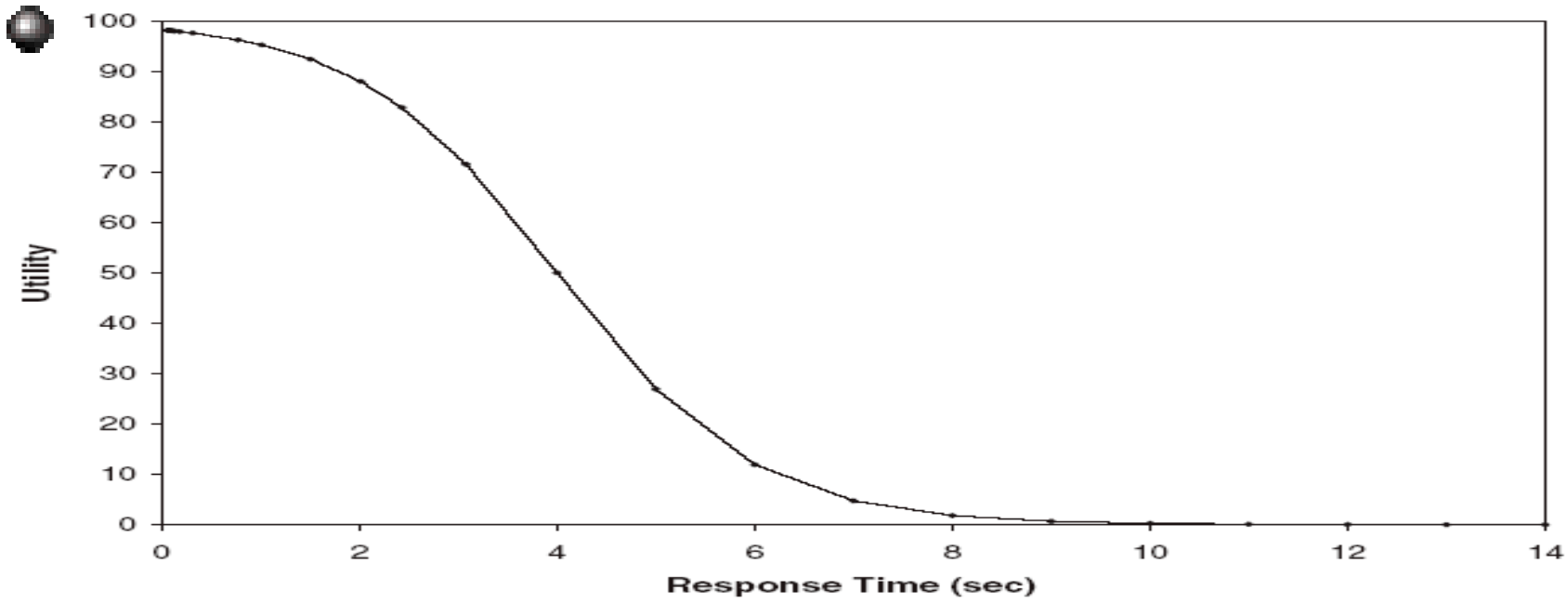


Figure 2. Utility as a function of response time.

Fig 3: Utility as a Function of Throughput

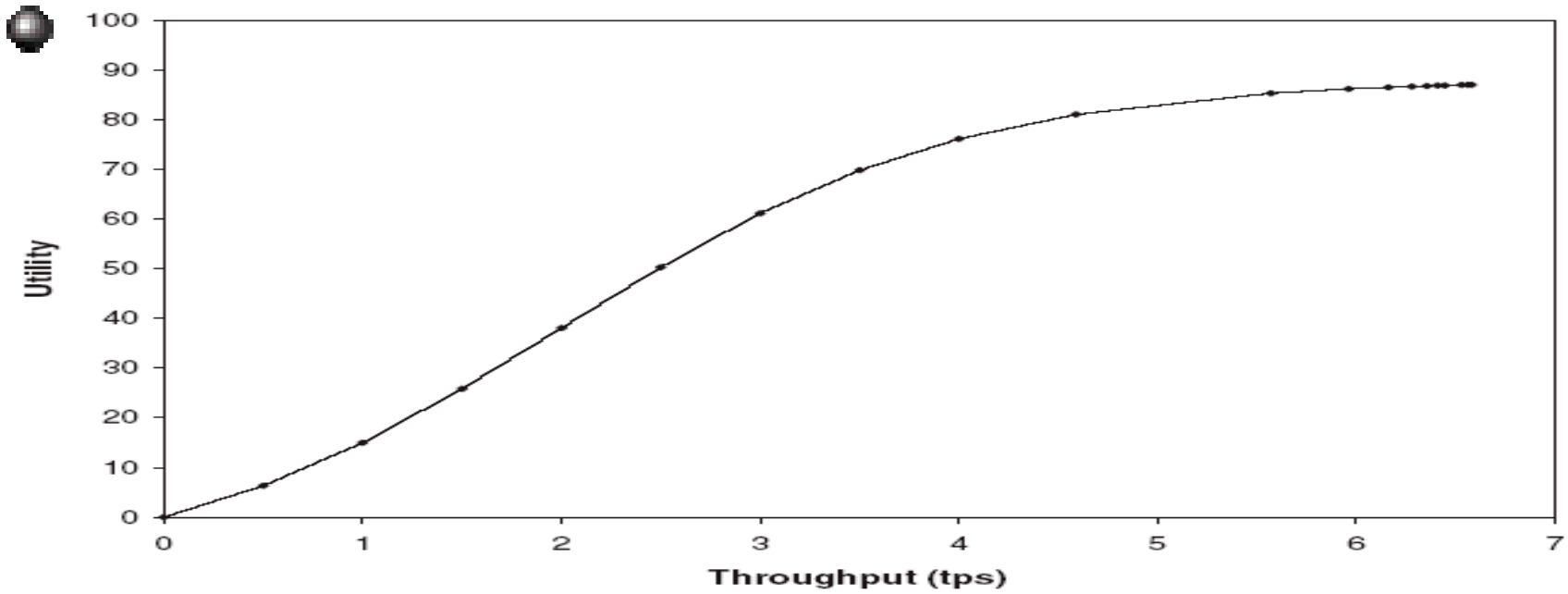


Figure 3. Utility as a function of the throughput.

Controller Approach: Local

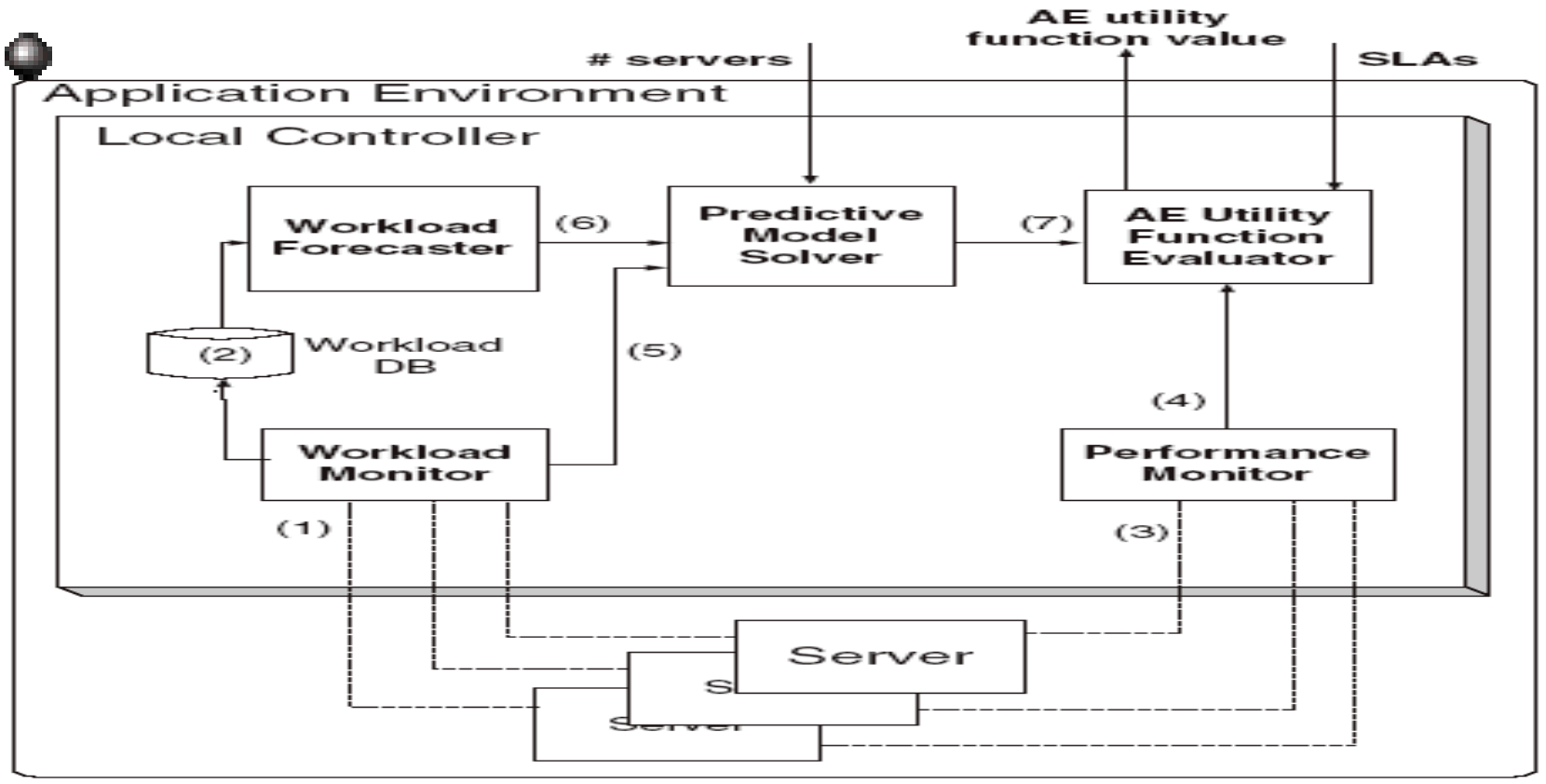


Figure 4. Local Controller.

Controller Approach: Global

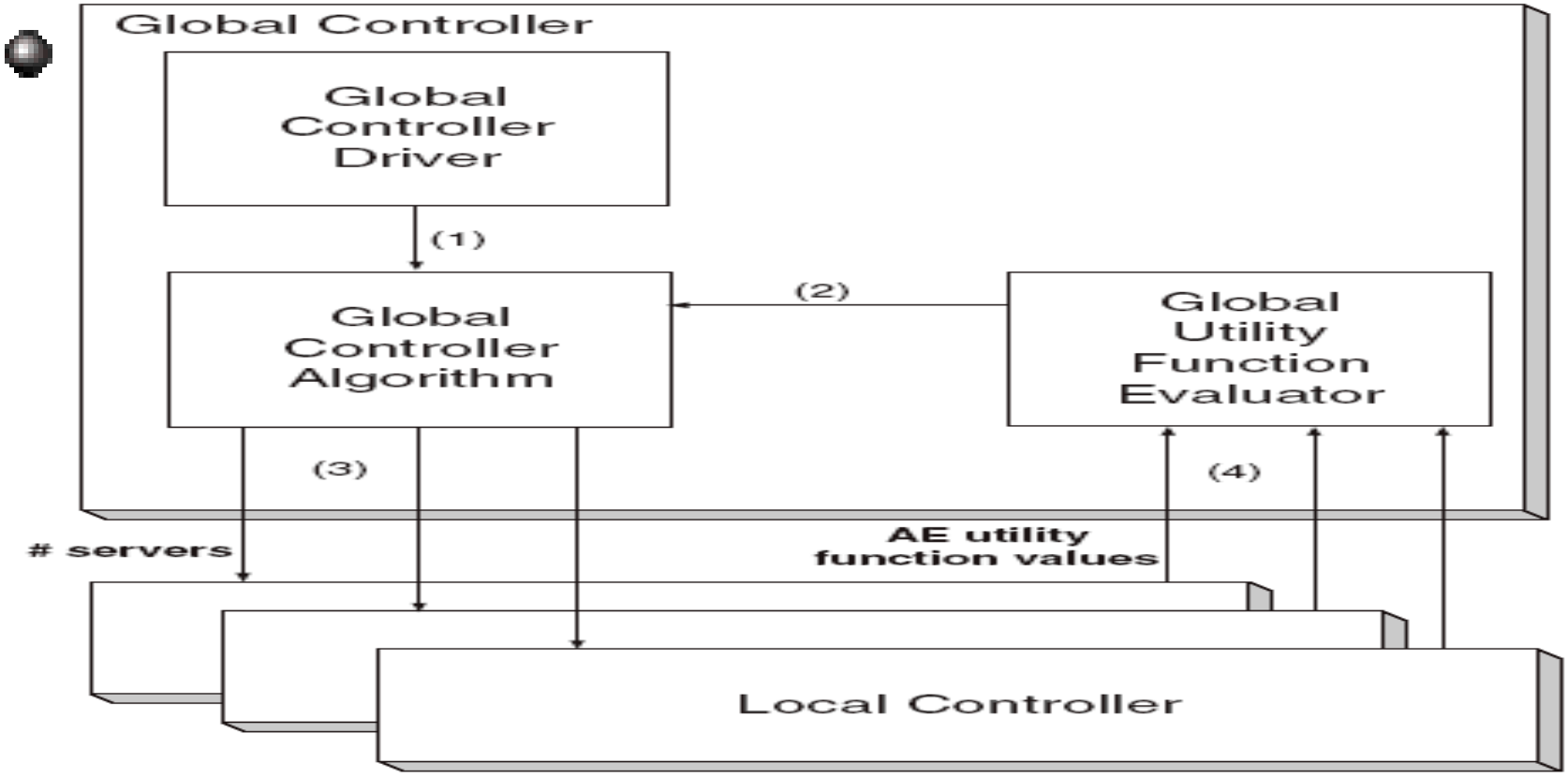


Figure 5. Global Controller.

Controller Approach: Controller Algorithm

LevelList$_0 \leftarrow \vec{n}_0$;
CandidateList $\leftarrow$ LevelList$_0$;
For i = 1 to d Do
  Begin
    LevelList$_i \leftarrow \emptyset$;
    For each $\vec{n} \in$ LevelList$_{i-1}$ Do
      LevelList$_i \leftarrow$ LevelList$_i \cup \mathcal{V}(\vec{n})$;
    LevelList$_i \leftarrow$ Top (k, LevelList$_i$);
    CandidateList $\leftarrow$ CandidateList $\cup$LevelList$_i$;
  End;
$\vec{n}_{opt} \leftarrow$ max (CandidateList)

Figure 6. Controller Algorithm.

- $\mathcal{V}(\vec{n})$: set of neighbors of configuration vector $\vec{n}$.
- LevelList$_i$: set of configuration vectors examined at level i of the beam search tree.
- CandidateList: set of all configuration vectors selected as the k best at all levels of the beam search tree.
- Top (k, $\mathcal{L}$): set of configuration vectors with the k highest utility function values from the set $\mathcal{L}$.
- $\vec{n}_0$: current configuration vector.

Performance Model for Online Transaction AEs

$$R_{i,s}(n_i) = \frac{D_{i,s}^{CPU}}{1 - \sum_{t=1}^{S_i} \frac{\lambda_{i,t}}{n_i} \times D_{i,t}^{CPU}} + \frac{D_{i,s}^{IO}}{1 - \sum_{t=1}^{S_i} \frac{\lambda_{i,t}}{n_i} \times D_{i,t}^{IO}}. \quad (9)$$

	Class		
	1	2	3
CPU	0.030	0.015	0.045
IO	0.024	0.010	0.030

Table 1. Service demands (in sec) for the example of Fig. 7

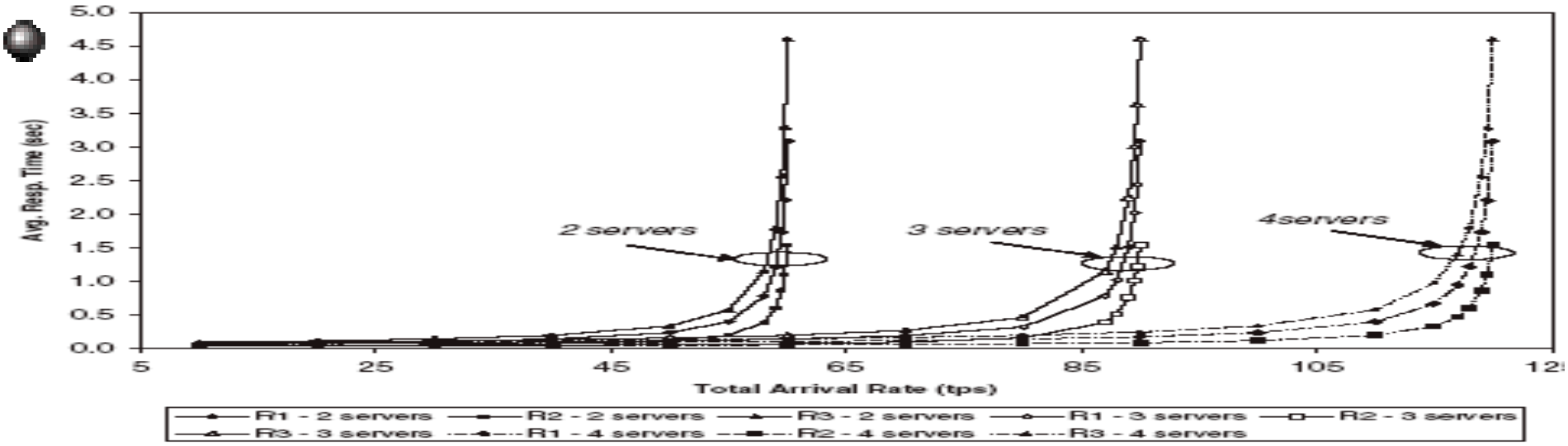
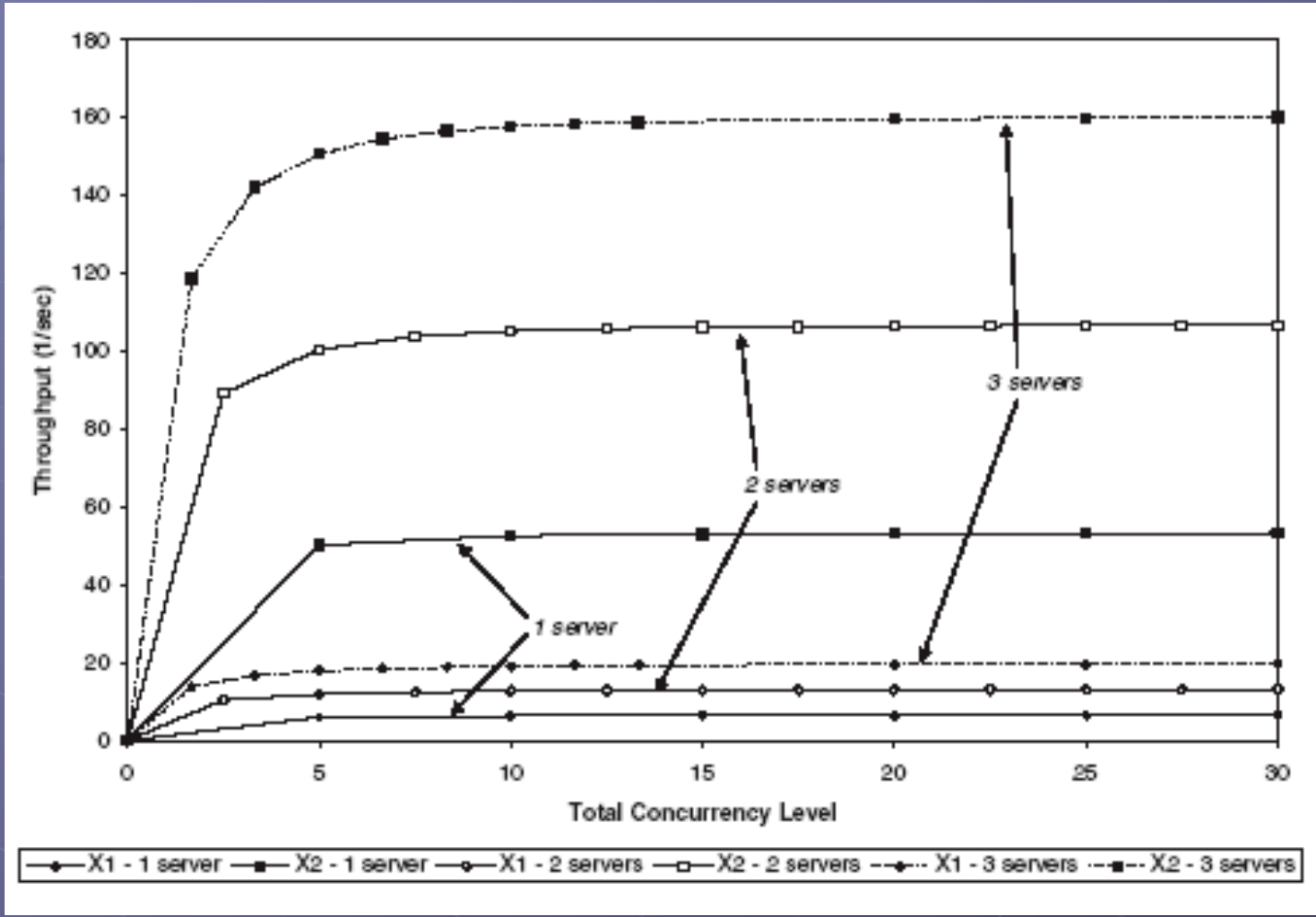


Figure 7. Response times for an Online AE.

Throughputs for a Batch Processing AE



	Class	
	1	2
CPU	0.030	0.015
IO	0.024	0.010

The Experimental Setting:
Simulated Data Center

- 3 AEs and 25 servers; 2 online AEs and 1 batch AE.
- Each AE runs multiple classes of transactions/jobs.
- The AEs were simulated using CSIM.
- Each simulated server has one CPU and one disk.
- The data center and all its AEs were simulated on one machine.
- The Controller code executed on separate computer - ran a beam search algorithm and used queuing network analytic models to decide the best allocation of servers to AEs.
- A local controller in each AE collects measurements from the AE (number of allocated servers, arrival rates per class for online AEs, and concurrency levels per class for batch AEs) and sends them to the global controller.
- The local controller is responsible for adjusting the number of servers deployed to each AE.
- Transactions for each class of the two online AEs are generated by separate workload generators with their own arrival rates. Separate workload generators for each class of each AE generate transactions for the online AEs.
- The batch AEs have as many threads per class as the concurrency level for that class.
- It is assumed that the switching cost is zero and is based on the fact that all applications are installed in all servers and that the switching cost to launch an application is ~ a few seconds.

Input Parameters

- The table shows:
  - The input parameters considered for each AE.
  - The service demands in seconds for each class
  - The SLA for each class (the $\beta i,s$ values),
  - The weights ($a_{i,s}$ values) used to compute the utility function *Ui* according to Eq. (7).
  - The values of β for the two online AEs are given in seconds because they correspond to response times.
  - The β values for the batch AE are given in jobs/sec because they correspond to minimum throughput requirements.
  - The table also shows the values of the concurrency levels for the three classes of *AE*3. These values do not change during the experiments.

Application Environment 1			
Type	Online		
S_1	3		
s	1	2	3
$D_{1,s}^{\mathrm{CPU}}$	0.030	0.015	0.045
$D_{1,s}^{\mathrm{IO}}$	0.024	0.010	0.030
$\beta_{1,s}$	0.060	0.040	0.080
$a_{1,s}$	0.350	0.350	0.300

Application Environment 2			
Type	Online		
S_2	2		
s	1	2	
$D_{2,s}^{\mathrm{CPU}}$	0.030	0.015	
$D_{2,s}^{\mathrm{IO}}$	0.024	0.010	
$\beta_{2,s}$	0.100	0.050	
$a_{2,s}$	0.450	0.550	

Application Environment 3			
Type	Batch		
S_3	3		
s	1	2	3
$D_{3,s}^{\mathrm{CPU}}$	0.005	0.010	0.015
$D_{3,s}^{\mathrm{IO}}$	0.004	0.008	0.012
$\beta_{3,s}$	400	250	150
$a_{3,s}$	0.350	0.350	0.300
$c_{3,s}$	30	20	10

Global Utility Functions and Scaling Factors

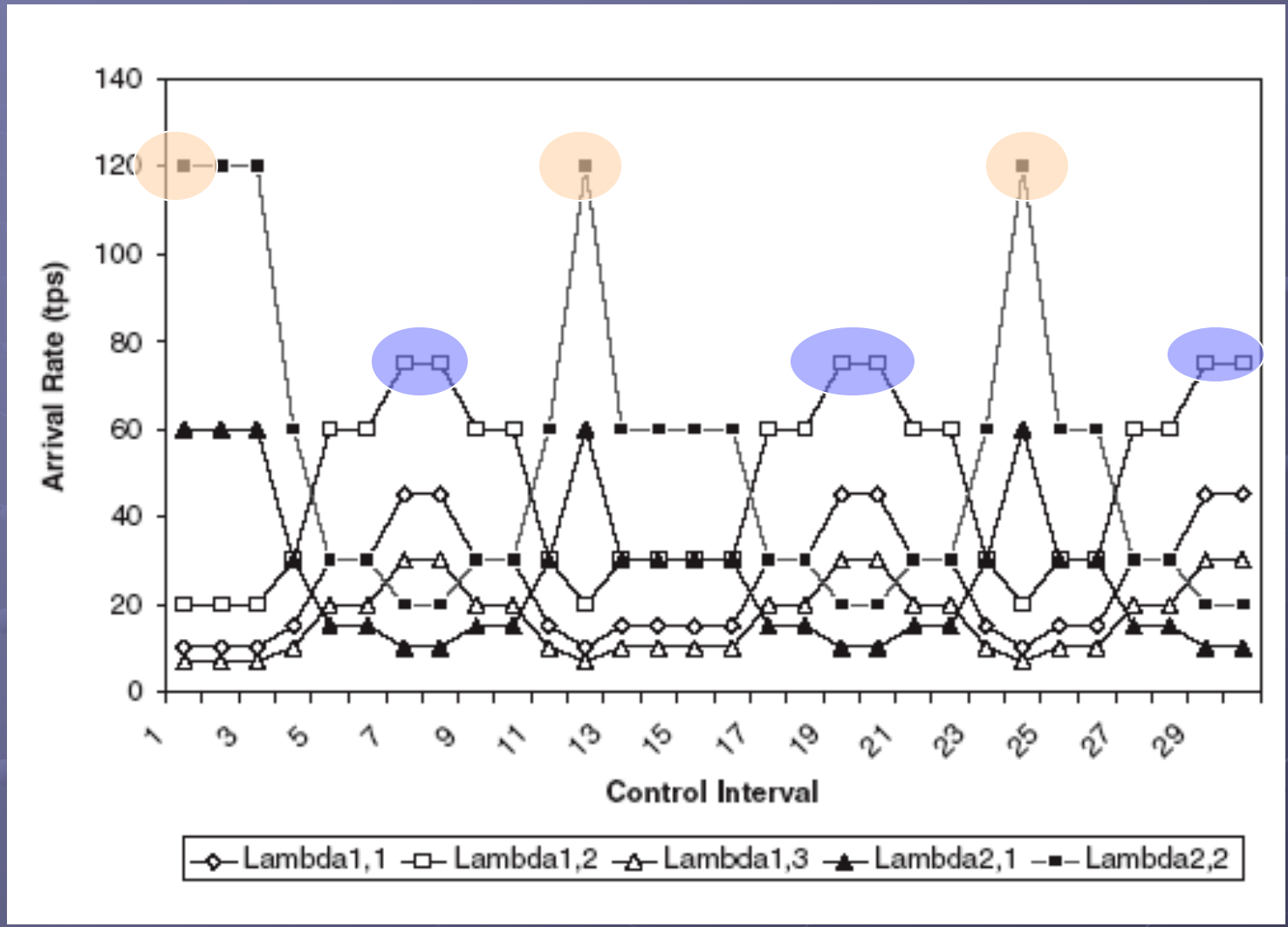
$$U_g = 0.35 \times U_1 + 0.35 \times U_2 + 0.3 \times U_3. \quad (10)$$

$$K_{i,s} = 100 \left(\frac{1 + e^{\beta_{i,s}}}{e^{\beta_{i,s}}} \right) \quad (11)$$

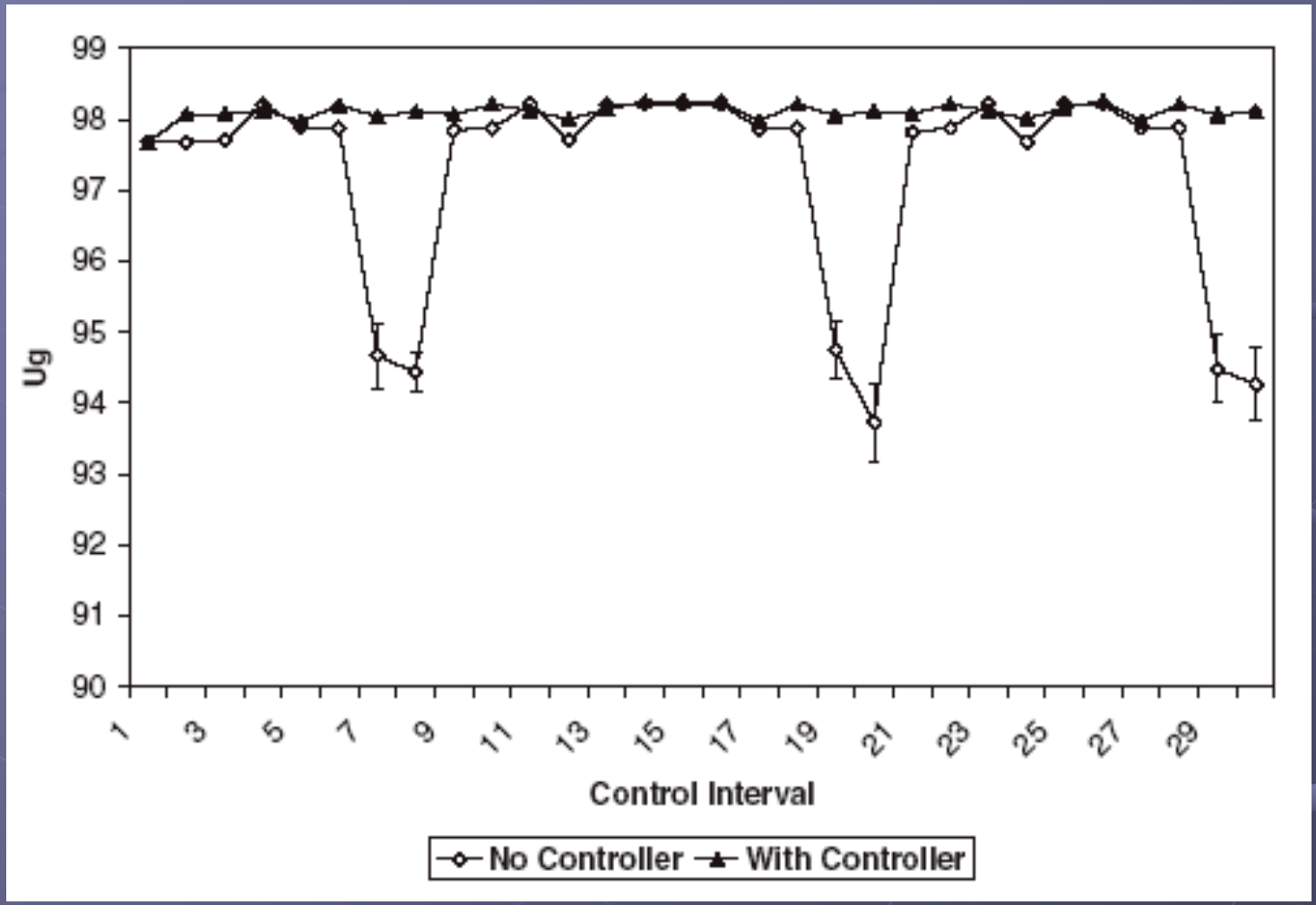
$$X^{+}_{i,s} = \frac{N - (M-1)}{\max(D^{\mathrm{CPU}}_{i,s}, D^{\mathrm{IO}}_{i,s})} \quad (12)$$

$$K_{i,s} = \frac{100}{\left(\frac{1}{1+e^{-x^{+}_{i,s}+\beta_{i,s}}} - \frac{1}{1+e^{\beta_{i,s}}} \right)}. \quad (13)$$

Variation of the Workload Intensity for AEs 1 and 2



Variation of the Global Utility Function



Variations in Local Utility Functions and the Number of Servers

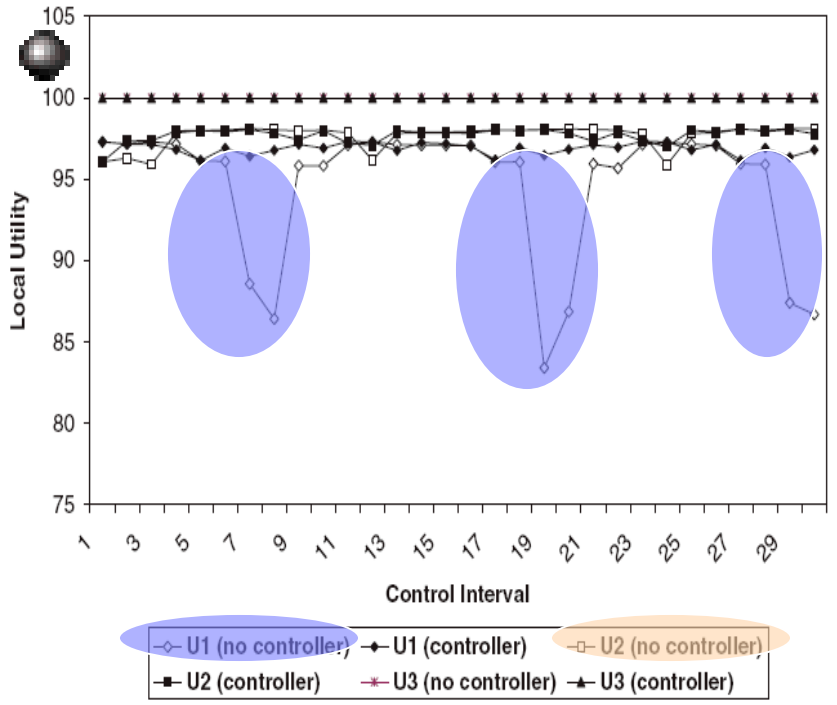


Figure 11. Variation of the local utility functions U_1, U_2, and U_3.

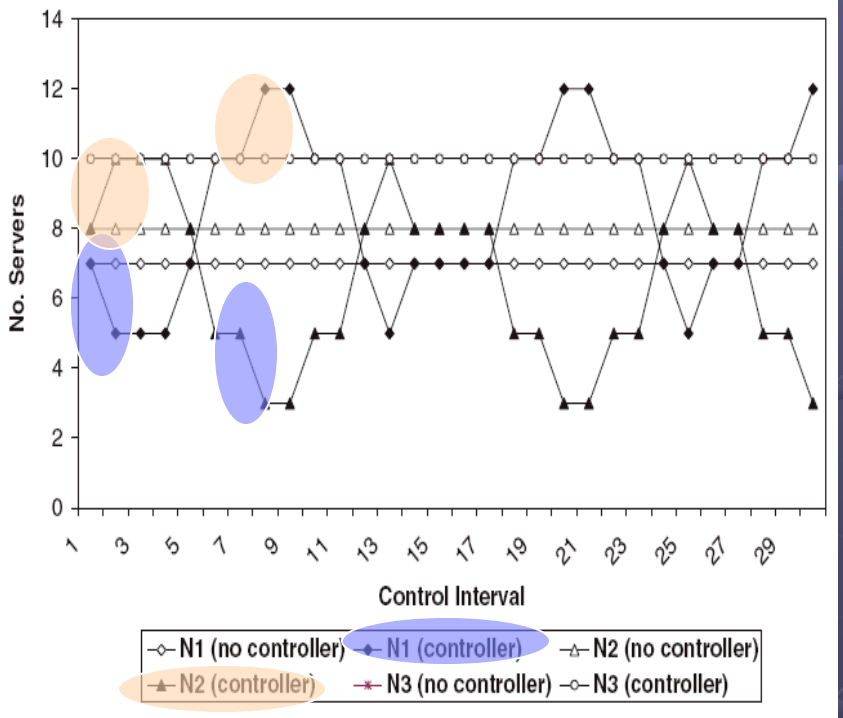
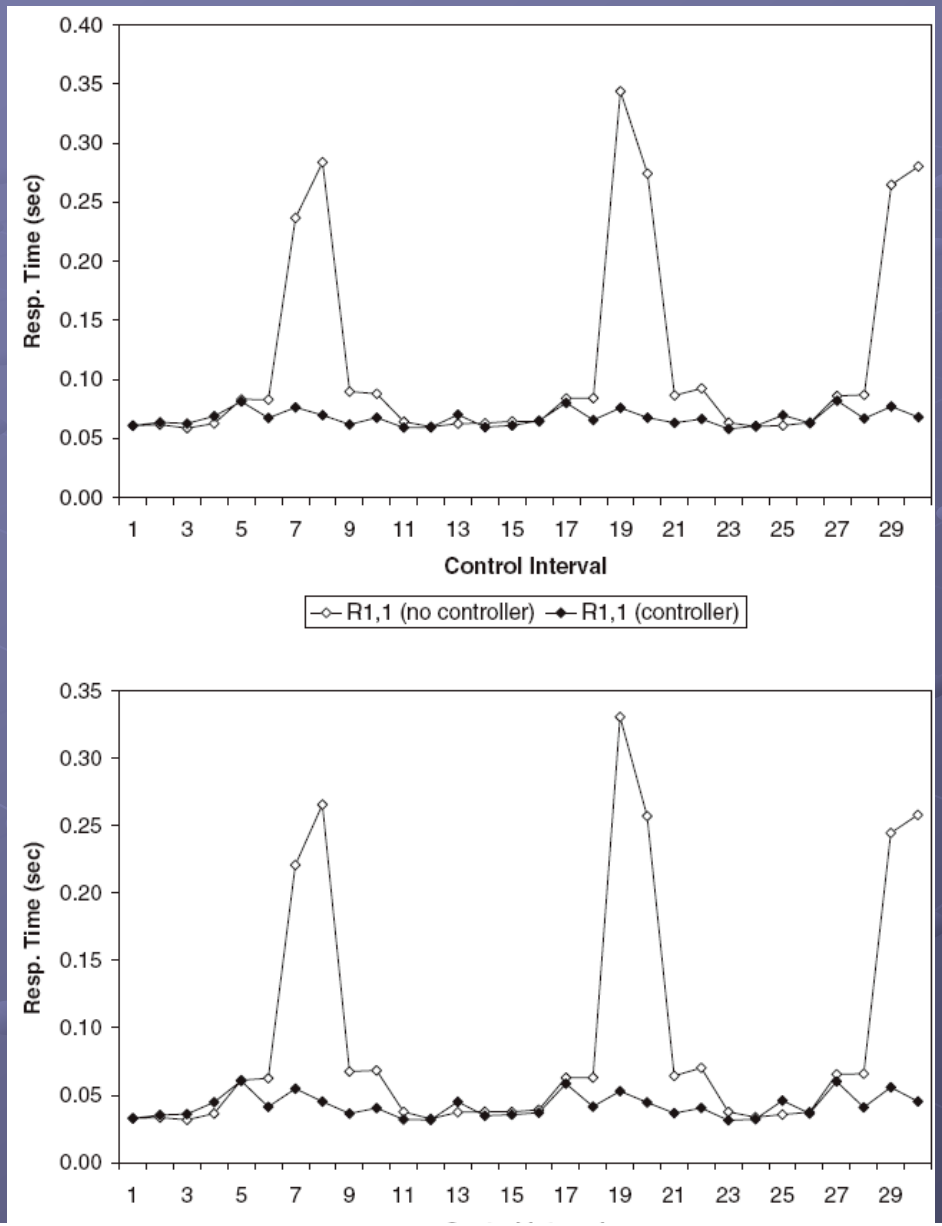


Figure 12. Variation of the number of servers n_1, n_2, and n_3.

Variation of the Avg. Response Times for Each Class Within AE_1



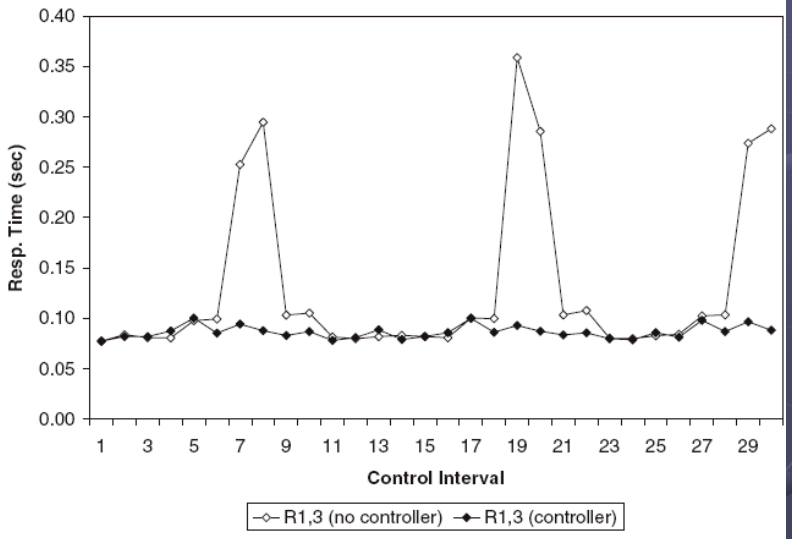
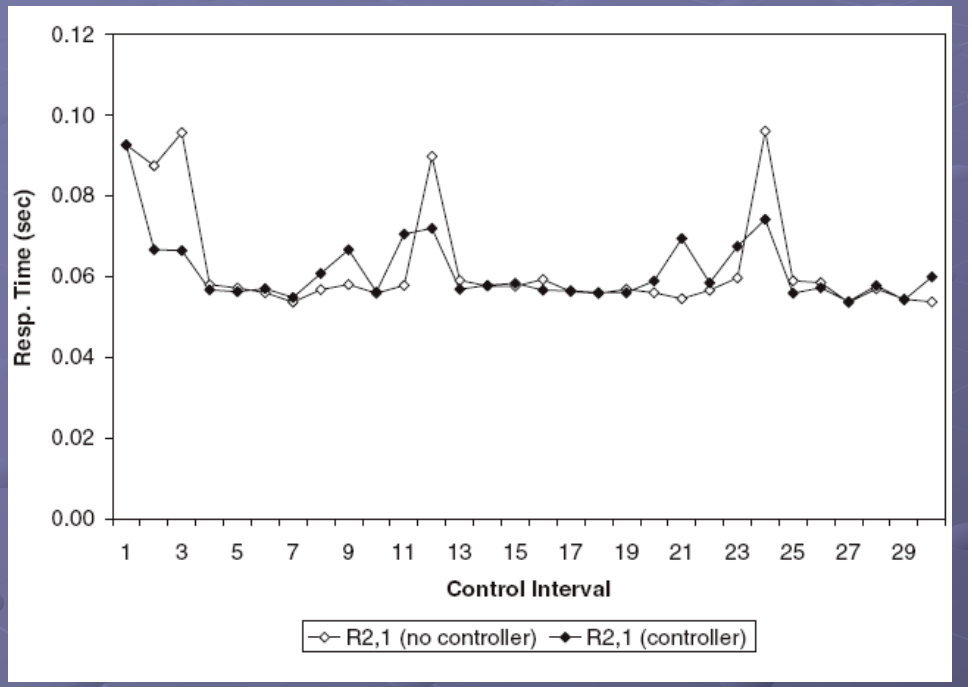


Figure 13. Variation of the average response times $R_{1,1}$ (top), $R_{1,2}$ (middle), and $R_{1,3}$ (bottom).

Variation of the Avg Response Times for AE_2



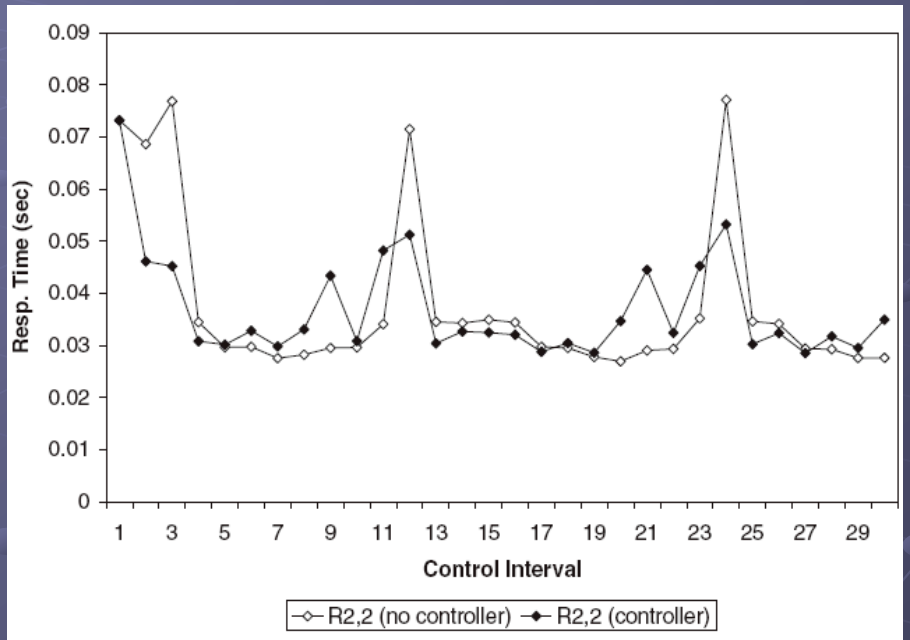
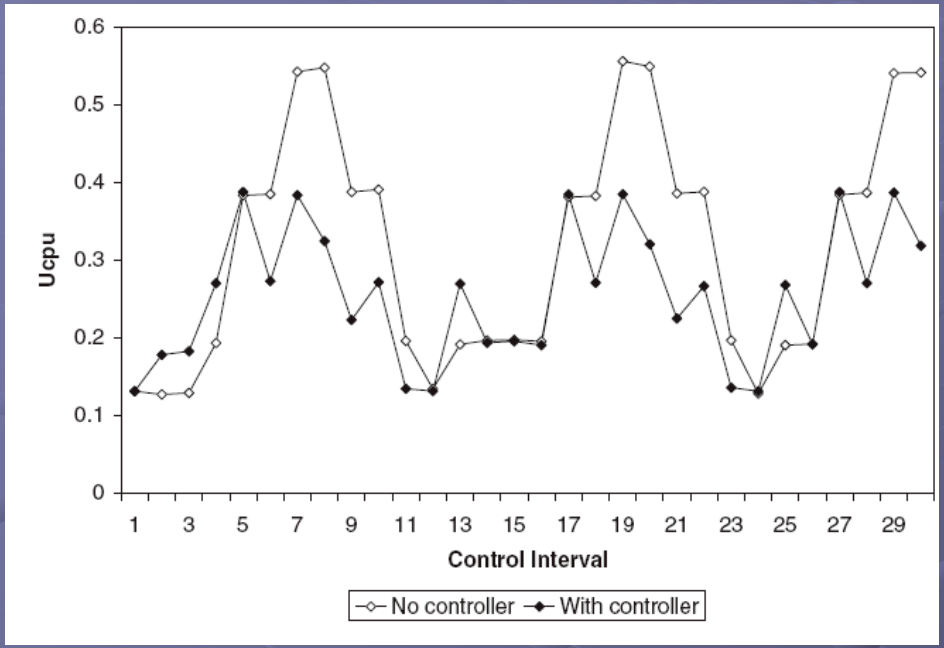


Figure 14. Variation of the average response times $R_{2,1}$ (top) and $R_{2,2}$ (bottom) for AE_2.

Utilization of the CPU and disk for AE_1



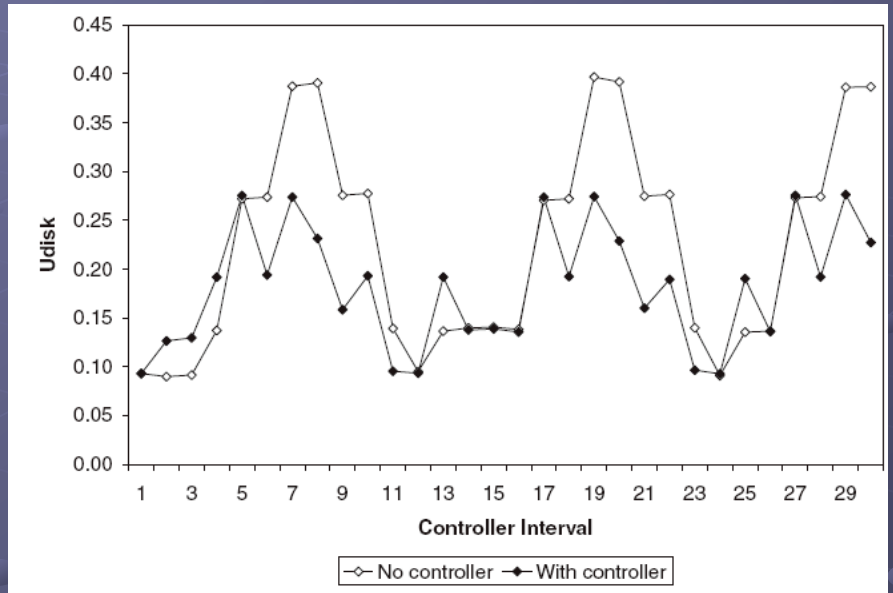
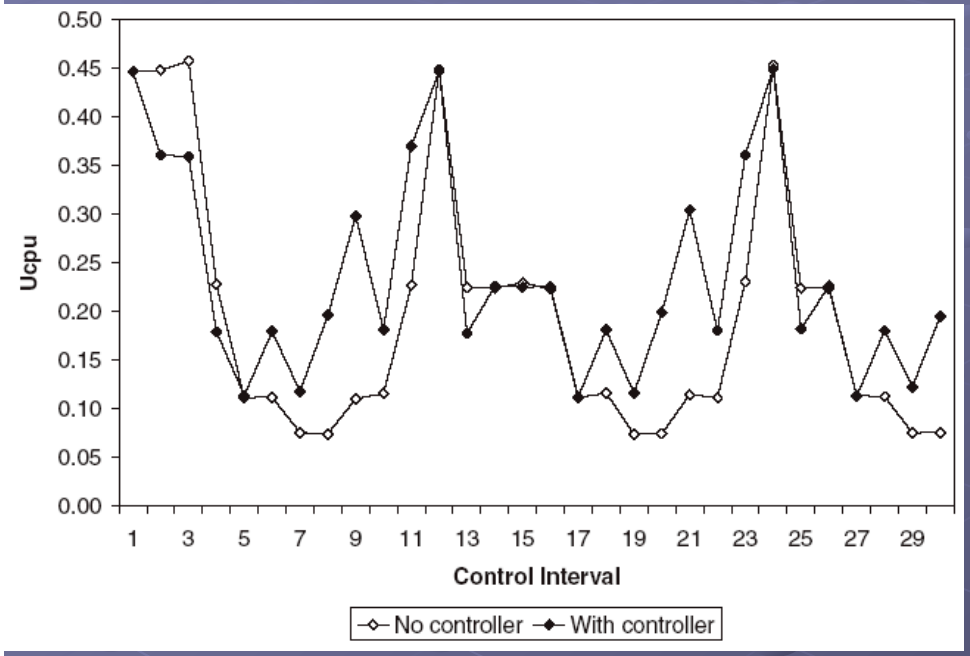
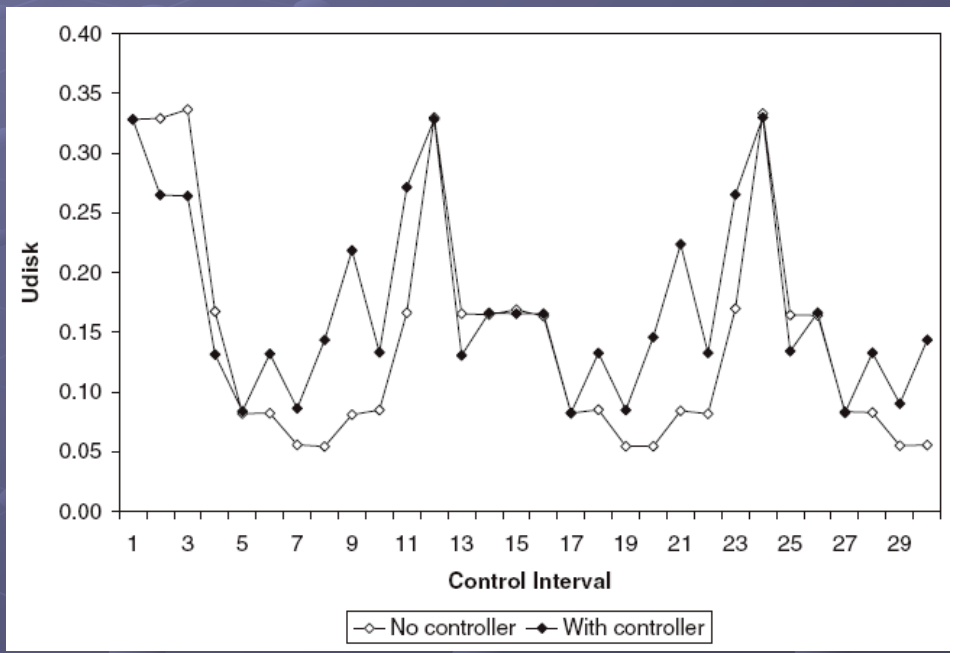


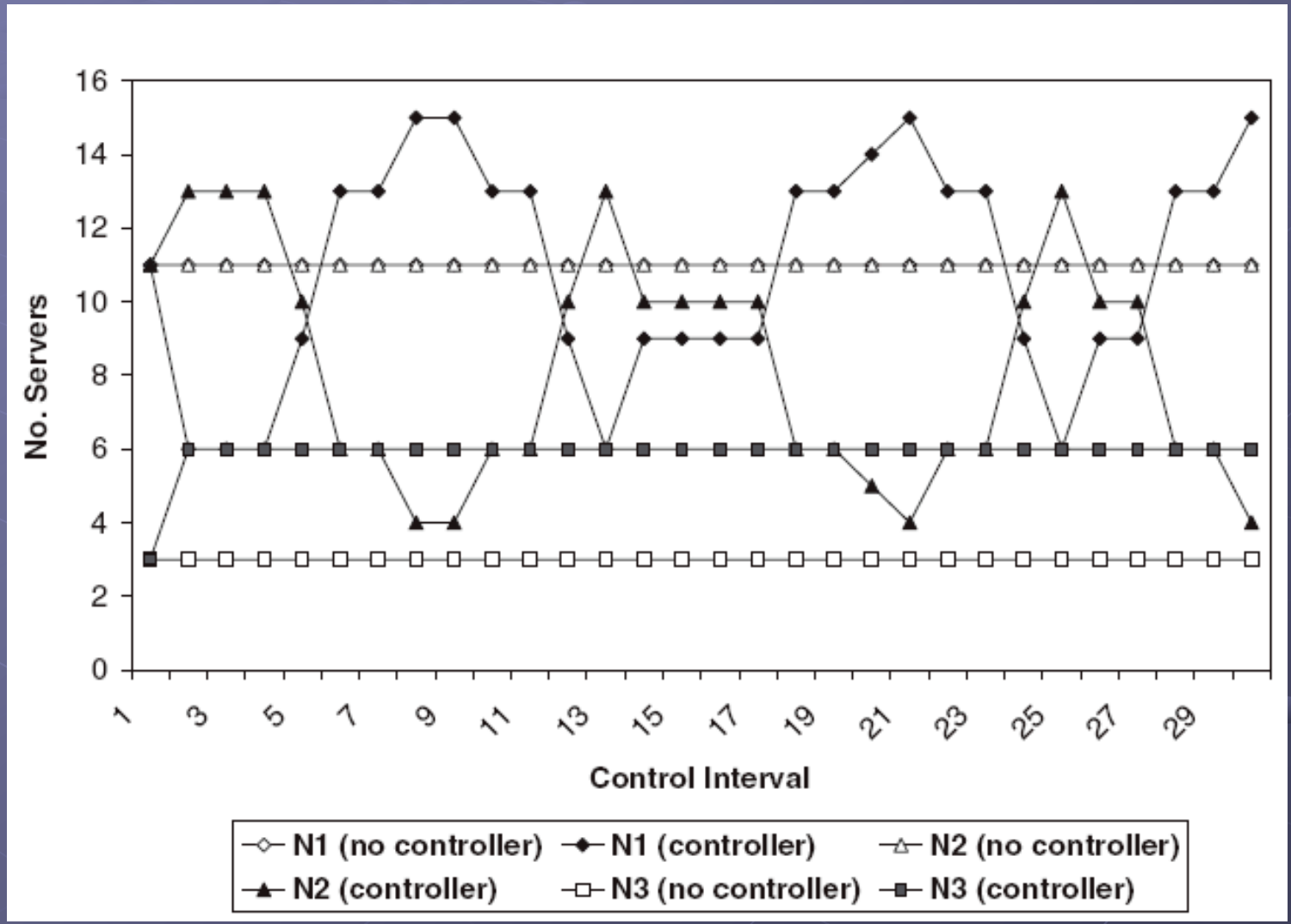
Figure 15. Utilization of the CPU and disk for AE_1.

Utilization of the CPU and disk for AE_2





Variation of the # of Servers With an Initial Allocation of 3 Servers to AE_3



Conclusion

- Analytic performance models can be efficiently used to design controllers that dynamically switch servers from one AE to another as needed.
- The approach is based on analytic performance models and not on simulation models.
- The approach scales well with the number of AEs, resources within an AE, and multiple transaction classes.
- The computational complexity of solving an ***open*** queuing network model for online AEs is $O(K\ S)$ where K is the number of resources (e.g, processors, disks) per server allocated to the AE and S is the number of transaction classes running on that AE **[11].**
- The computational complexity of solving ***closed*** queuing network models for batch AEs is $O(I\ K\ S2\ )$ where K and S are as before and I is the number of iterations of AMVA **[11].**
- In most cases, AMVA tends to converge very fast (in less than 100 iterations) for tolerance values of $10-5$.
- Configurations can be evaluated quickly.
- The number of configurations to be evaluated each time the controller runs is a function of the breadth and depth parameters used in the beam search technique.