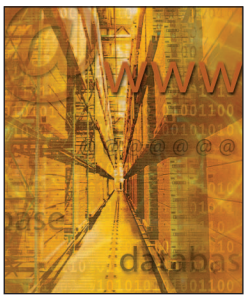

Allocating Applications in Distributed Computing

Daniel A. Menascé • *George Mason University*

We can use grid computing[1] and virtualization[2] to distribute computing applications on the Internet, letting a collection of interconnected servers host a set of applications to balance the processing load and improve performance. An important question is which applications to assign to which servers so we can meet the applications' quality of service (QoS) goals, given the servers' bandwidth and capacity constraints. In this column, I precisely define the problem and discuss a solution through a motivating numerical example.

Basic Model

Consider a set $A = \{a_1, ..., a_M\}$ of M applications and a set $S = \{s_1, ..., s_K\}$ of K servers. The bandwidth of the network connection for each server s_j in S is B_j bits per second. We can deploy more than one application to any server, but a given application runs entirely on a single server (see Figure 1). Applications 1 through 3 share server 1's resources, including its bandwidth; applications 4 and 5 share server 2; and server 3 is dedicated to running application 6. Contention for server resources increases as more applications are deployed to a server, degrading each application's performance. The number of possible allocations of servers to applications grows fast with M and K. In fact, the number of all possible allocations is equal to K^M because we can independently allocate each of the M applications to one of the K servers. There are 729 (= 3^6) possible allocations for the example in Figure 1. However, not all possible allocations are feasible due to bandwidth and QoS constraints.

QoS goals are associated with each application. Such goals for application a_i include its maximum average execution time $r_i^{\max}$ and its minimum throughput $x_i^{\min}$. The workload intensity associated with application a_i is equal to λ_i requests per second. The resource demand of application a_i is characterized by the tuple (p_i, d_i, n_i), where p_i is the application's processing time at one of the servers in S, taken as a reference server; d_i is the application's I/O demand measured in seconds of disk service time; and n_i is the application's network demand measured in bps. Let's consider that all servers have the same capacity. Relaxing this condition is pretty straightforward.

Not all possible allocations are feasible. If the network demands of the applications allocated to a given server exceed that server's bandwidth, for example, the allocation isn't feasible. We express this constraint as

$$B_j \geq \sum_{a_i \in \Omega(s_j)} n_i \quad j = 1, ..., K, \tag{1}$$

where $\Omega(s_j)$ is the set of applications allocated to server s_j.

An allocation isn't feasible if at least one of the applications' QoS goals is violated. Let r_i be the average response time of application a_i. Then, for any feasible allocation, the following relationship must hold:

$$r_i \leq r_i^{\max} \qquad i = 1, ..., M. \tag{2}$$

The average response time r_i of application a_i depends on its resource demands, its workload intensity, and the resource demands p_i and d_i and workload intensities of all applications allocated to the same server as a_i. We can use a multiclass open queuing network model[3] to compute the average response time for each application in a given allocation. To do so, we must first create a queuing network model for each server, with the applications allocated to that server for a given allocation constituting the model's var-

 Published by the IEEE Computer Society 1089-7801/05/$20.00

ious classes. In the case of Figure 1, for instance, we must solve three queuing network models, one for each server. The model for server 1 has three classes, one for each of the three applications allocated to that server.

Numerical Example

Let's look at an example with two servers and four applications: it has 16 (= 2^4) possible allocations. Table 1 shows the four applications' processing, disk, and network demands, as well as their workload intensity and maximum response times. The servers have the following bandwidths: B_1 = 100,000 bps and B_2 = 150,000 bps.

Table 2 shows all 16 allocations. The first four columns show the allocations of the two servers to the four applications, and the next two columns indicate each allocation's resulting bandwidth requirements. Equation 1 compares these requirements to the server bandwidth. If the bandwidth requirement due to an allocation exceeds the server's bandwidth, the allocation isn't feasible, which column seven indicates.

We compute the average response time for each application only for the allocations that pass the bandwidth feasibility test and record the results in columns 8 through 11. We can then compare these response time values for each application to the respective maximum response times shown in Table 1 (see Equation 2). The last column of Table 2 shows that only two allocations are feasible at this point: place applications 1 and 3 in server 1 and applications 2 and 4 in server 2; or place application 1 on server 1 and the other three applications on server 2. We could use other criteria (such as cost or availability) to select one of these two allocations.

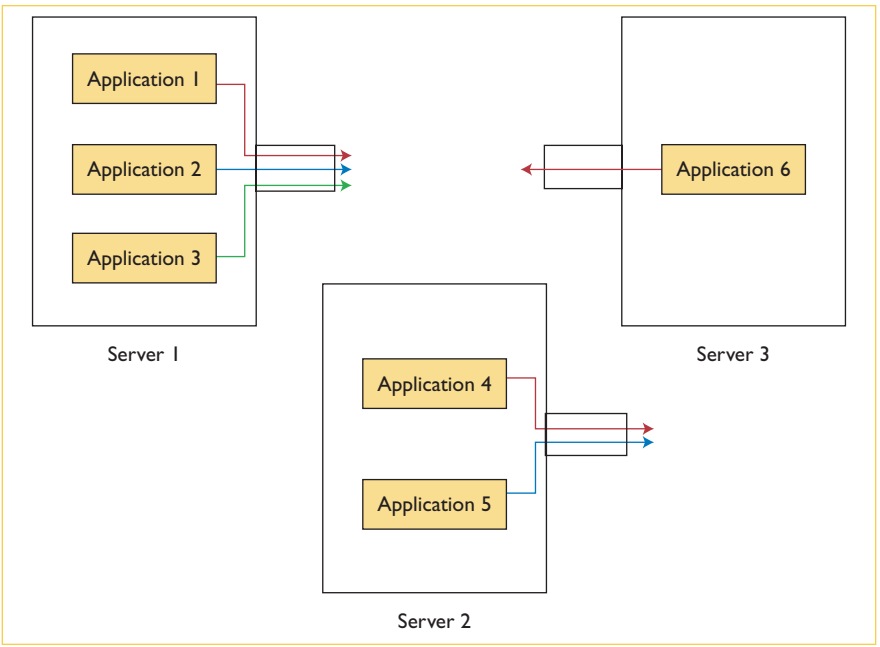

Figure 1. Example of application allocation to servers. Applications 1 through 3 share the resources of server 1, applications 4 and 5 are allocated to server 2, and server 3 is dedicated to running application 6.

The analysis I just described consists of three steps:

1. Enumerate all possible allocations.
2. Eliminate the allocations that aren't feasible due to the bandwidth constraint.
3. Compute the average response time for each application for the allocations remaining from step 2 and eliminate those that violate the response time constraint.

We can optimize step 1 by avoiding the enumeration of all possible allocations. First, order all applications in decreasing order of network demand. Thus, application 1 is the one with the highest network demand, and application M is the one with the lowest. Start assigning servers to applications from a_1 to a_M. Then, after each allocation, update the total required bandwidth for the server just allocated. If it exceeds the server's bandwidth, stop that allocation and ignore all allocations with the same prefix. Consider the numerical example just described,

Table 1. Application characteristics.

Applications	Processing demand (sec)	Disk demand (sec)	Network demand (bps)	Workload intensity (tps)	Maximum response time (sec)
1	0.7	0.9	50,000	0.4	3.0
2	0.3	0.45	25,000	0.3	1.2
3	0.2	0.18	30,000	0.4	1.0
4	0.25	0.35	80,000	0.55	1.0

Table 2. List of allocations and feasibility analysis.

Application				Bandwidth Requirements			Response time (sec)				
1	2	3	4	Server 1 bandwidth (bps)	Server 2 bandwidth (bps)	Feasible bandwidth?	1	2	3	4	Feasible response time?
1	1	1	1	185,000	—	No					
1	1	1	2	105,000	80,000	No					
1	1	2	1	155,000	30,000	No					
1	1	2	2	75,000	110,000	Yes	2.89	1.37	0.50	0.80	No
1	2	1	1	160,000	25,000	No					
1	2	1	2	80,000	105,000	Yes	2.68	1.06	0.63	0.84	Yes
1	2	2	1	130,000	55,000	No					
1	2	2	2	50,000	135,000	Yes	2.38	1.18	0.59	0.94	Yes
2	1	1	1	135,000	50,000	No					
2	1	1	2	55,000	130,000	Yes	3.21	0.93	0.47	1.21	No
2	1	2	1	105,000	80,000	No					
2	1	2	2	25,000	160,000	No					
2	2	1	1	110,000	75,000	No					
2	2	1	2	30,000	155,000	No					
2	2	2	1	80,000	105,000	Yes	3.35	1.58	0.78	0.72	No
2	2	2	2	—	185,000	No					

but with the applications sorted as 4, 1, 3, 2 in decreasing order of bandwidth demand. For example, all allocations that start with 1, 1 (that is, server 1 allocated to applications 4 and 1) aren't feasible because the required bandwidth on server 1 would be 130,000 bps, which exceeds its bandwidth of 100,000 bps. Thus, we don't need to consider allocations such as 1, 1, 2, 1 or 1, 1, 2, 2. This observation can significantly reduce the number of allocations we need to analyze.

We might have to consider other issues when deciding where to run applications in a distributed system, including the cost of running an application on a given machine, the cost of shipping the data an application needs to a given machine, each machine's availability, and an application's affinity for a given machine's architecture. Taking all these factors into account might significantly increase the complexity of the application-allocation problem. If the number of allocations to consider is too high, we might think about using heuristics that don't cover all possible allocations but don't necessarily find the optimal allocation. □

Acknowledgment

Grant number NMA501-03-1-2022 from the US National Geospatial-Intelligence Agency (NGA) partially supports this work.

References

1. I. Foster and C. Kesselman, "The Grid in a Nutshell," *Grid Resource Management*, J. Nabrzyski, J.M. Schopf, and J. Weglarz, eds., Kluwer Academic, 2004, pp. 3–13.
2. M. Milenkovic et al., "Toward Internet Distributed Computing," *Computer*, vol. 36, no. 5, 2003, pp. 38–46.
3. D.A. Menascé, V.A.F. Almeida, and L.W. Dowdy, *Performance by Design: Capacity Planning by Example*, Prentice Hall, 2004.

Daniel A. Menascé is a professor of computer science, codirector of the E-Center for E-Business, and director of the MS in E-Commerce program at George Mason University. He received a PhD in computer science from UCLA. Menascé is a fellow of the ACM and a recipient of the A.A. Michelson Award from the Computer Measurement Group. Contact him at menasce@cs.gmu.edu.