A DYNAMIC PROGRAMMING MODEL OF TWO HETEROGENEOUS CLUSTERS WITH CO-ALLOCATION OF JOBS

D. FILIPPOPOULOS AND H. KARATZA

Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki, 541 24, Greece
E-mail : {fildim, karatza}@csd.auth.gr

Abstract: We focus on the problem of the co-allocation of rigid parallel jobs in multi-cluster or Grid systems. Co-allocation is the simultaneous allocation of nodes that belong to different clusters to the tasks of the same parallel job. We construct a dynamic programming model, as simple as needed to be tractable, and compute the optimal policy with respect to whether to co-allocate the next job to start execution or not. The optimal policy is tabulated and compared to the one that does not employ co-allocation at all. Our results indicate that co-allocation can be beneficial even when the communication speed between different clusters is slow compared to within the same cluster. Additionally, we propose a heuristic policy and compare its performance with optimal policy by simulation. We find that our heuristic policy performs close to optimal for a wide range of parameters.

Key Words: Co-allocation, Dynamic Programming, Optimal Policy ....

1 INTRODUCTION

Over recent years, heterogeneous clusters of servers have been put together to form the so called *multi-cluster systems* or computational *Grids*. Multi-clusters support a variety of users and services. The idea of combining clusters of commodity workstations has been proved to be extremely effective in delivering computational power at low cost. Users are not necessarily aware of the internal structure of the system. Their main concern is to receive the results of their submission with as little delay as possible.

Therefore, efficient resource management policies are imperative, so that the resources of the computational Grid are fully utilized and users experience as short delays as possible. However, in real systems, this has proved to be a difficult task. This is because the workload is not generally predictable. For example, in [Riska et al, 2002], it is stated that a clustered web server can experience overload periods due to the announcement of a special event (the results of the World Cup Soccer 1998). In addition, in [Karatza, 2002] a distributed system with time varying workload is studied for different task scheduling policies. It is shown by simulation that the choice of a scheduling policy is important to achieve good overall performance.

One possible alternative to efficiently scheduling parallel jobs in multi-cluster systems, is to employ *co-allocation*. Instead of scheduling jobs locally within the cluster they were initially submitted to, tasks or components of parallel jobs are scheduled in different clusters. Although such a decision would result in longer delays for the co-allocated job, since inter-cluster communication is much slower than intra-cluster communication, the co-allocated job may receive the benefit of starting execution sooner. Furthermore, the fragmentation of the system can be reduced if such a decision is taken at selected instants.

The purpose of this paper is to examine if co-allocating jobs can be beneficial and, if so, when such a decision should be taken. We propose a dynamic programming model, as simple as needed to ensure tractability, and compute the optimal policy for the resulting optimization problem. To our knowledge, such a model has not been presented in the literature so far.

We review related research. A substantial contribution is in [Bucur and Epema, 2003]. There, simulation is used to evaluate all configurations of

a multi-cluster system. Jobs are separated in four types with respect to their structure. In terms of co-allocation, the authors define a metric called *Communication Speed Ratio*, which is equal to the ratio between the intra-cluster and inter-cluster communication speed. They study the benefits of co-allocating jobs when there is a choice, and their results indicate that co-allocating jobs is a very good choice for systems with speed ratios that are not extremely large. Specifically, they claim that at low utilizations of the system avoiding co-allocation may be a good choice, while at high loads co-allocation is significantly better, even with speed ratios larger that 50.

In [Ernemann et al, 2002] a multi-cluster system is studied for different workloads derived form real traces. The system consists of a central scheduler that allocates parallel jobs to clusters. All cluster have the same local scheduling policy. Co-allocation of jobs (called multi-site scheduling in that paper) is compared to load balancing as well as to a system where clusters work independently of each other. To model the impact the communication delays to the execution times of co-allocated jobs, the authors define a factor by which the total execution times of co-allocated jobs are multiplied. An important result is that co-allocating jobs, instead of keeping them local to a single cluster, is beneficial for values of the factor less than or equal to 1.25.

In [Palmer and Mitrani, 2003a], a system consisting of 2 clusters of servers is considered. Jobs are all 1-component, e.g. they need exactly one free server to start execution. Holding costs are assumed in each of the 2 clusters. A decision maker can switch a server from one cluster to the other. This needs time to be done, during which the switched server is unavailable. Additionally, a cost is payed for each switch. The optimal policy for a truncated version of state space is computed and compared with heuristic ones. [Palmer and Mitrani, 2003b] is a generalization of [Palmer and Mitrani, 2003a] with more than 2 clusters and jobs types. Their results indicate that allocating nodes from one cluster to another is beneficial if it is done dynamically. Another paper that uses the same methodology is [Martin and Mitrani]. In that paper, the authors investigate how to best route jobs among two queues whose servers are subject to breakdowns and repairs. The optimal routing policy is computed and compared to heuristic ones. A heuristic policy that performs to close to optimal for a wide range of parameters is introduced. In addition, the policy that routes each job to the shortest of the two queues (it is called *shortest queue* routing in that paper) performs very well.

Our approach follows the approach and rationale of [Palmer and Mitrani, 2003a,b]. However, we are considering 2-component jobs as well, and no switches occur. Furthermore, our model involves only a small number of servers in each cluster. This is because the presence of 2-component jobs imposes an additional complexity in the model that can be tackled by reducing the number of servers in each of the clusters.

The rest of the paper is organized as follows: In section 2, the model is described in detail. In section 3, the technique used for the computation of the optimal policy is reviewed and in the section after that, performance and complexity issues are discussed. In section 5, we select specific values for the parameters of our model and compute the optimal policy. In section 6, simulation is used to compare the optimal policy with the local and a heuristic one, and in section 7, general conclusions are stated. In the last section discussion for future research is included.

2 DESCRIPTION OF THE MODEL

The model used in our analysis is depicted in figure 1. The system consists of 2 heterogeneous clusters: 1 and 2. Cluster 1 consists of 2 nodes (or servers) and a single unbounded queue. Cluster 2 consists of a single node and an unbounded queue as well. Jobs arrive to cluster $i(i$=1,2) following an independent Poisson process with rate λ_i. Each job submitted to cluster $i(i$=1,2) incurs a cost c_i per unit of time it spends in the system.

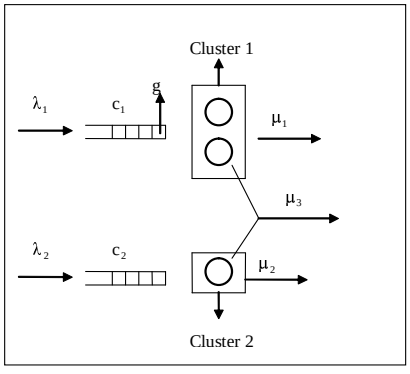

Figure 1: A queueing system with two heterogeneous clusters. Jobs submitted to cluster 1 can be co-allocated (service rate μ_3) or executed locally (service rate μ_1). Jobs submitted to cluster 2 are local jobs (service rate μ_2).

Jobs are characterized by the execution or service time and the number of nodes they require. A job may require 1 node (1-component job) with probability p or 2 nodes (2-component job) with probability $1-p$. The number of nodes a job requires is specified by the user at submission time and it is a single fixed number. It does not change during execution. Furthermore, jobs acquire all the nodes they need at the same time and release them, simultaneously, as soon as they finish execution. Such jobs are called *rigid* [Bucur and Epema, 2003]. We assume that jobs submitted to cluster 2 are 1-component only while those submitted to cluster 1 may have 1 or 2 components. Service times for jobs submitted to cluster i(i=1,2) and executed locally follow an exponential distribution with rate μ_i, independent of the number of nodes they require. Co-allocated jobs are executed at exponentially distributed time with rate μ_3. Additionally, the FCFS scheduling discipline is used at both queues.

A job at head of queue at cluster 1 can be co-allocated or not. A job can be co-allocated if it is a 2-component job, there is at least one free node at cluster 1 and there is no other co-allocated job. In that case, it occupies a node from cluster 1 and the unique node of cluster 2. The job receiving service at cluster 2 (if any), is preempted and checkpointed. It resumes service from the point of the last interruption, as soon as the co-allocated job finishes execution. Alternatively, a job at cluster 1 can be scheduled locally, e.g. no co-allocation is applied. Then, if the number of free nodes at cluster 1 is greater than or equal to those required by the job, it can start execution immediately. Otherwise, it remains in the queue.

Decisions are applied instantaneously, just after an arrival or job completion instant. To simplify our model, we assume that at most 1 job from queue at cluster 1 can start execution at each decision epoch. In the rest of the paper, number 1 corresponds to action *"Co-allocate"* job and 0 corresponds to *"Do not co-allocate job"*.

The system state $\overrightarrow{S}$ is defined as $\overrightarrow{S} = (q_1, q_2, g, x_1, x_2, x_{11})$, where q_1 is the number of jobs waiting to enter service at queue 1, q_2 is the queue length at cluster 2, including the one in service, g is the number of nodes required by the job (if any) at head of queue at cluster 1, x_1 is the number of 1-component jobs receiving service at cluster 1, x_2 is the number of 2-component jobs receiving service at cluster 1, and x_{11}, is the number of jobs that are co-allocated. The parameter g takes three values: 0, 1, 2. Value 0 corresponds to the case when there are no jobs waiting in queue of cluster 1, while, values 1 and 2 are assigned to g when the job at head of queue at cluster 1 requires 1 and 2 nodes respectively. Thus, $q_1 = 0 \Leftrightarrow g = 0$. The set Λ of all valid states is given by $\Lambda = \Lambda_1 \bigcup \Lambda_2$, where, $\Lambda_1 = \{(0, q_2, 0, x_1, x_2, x_{11}) : q_2 \geq 0,\ x_1 \geq 0,\ x_2 \geq 0,\ 0 \leq x_{11} \leq 1,\ x_1 + 2x_2 + x_{11} \leq 2\}$ and, $\Lambda_2 = \{(q_1, q_2, g, x_1, x_2, x_{11}) : q_1 \geq 1,\ q_2 \geq 0,\ x_1 \geq 0, x_2 \geq 0, 0 \leq x_{11} \leq 1, 1 \leq g \leq 2, x_1 + 2x_2 + x_{11} \leq 2\}$.

Supposing the process enters state $\overrightarrow{S} = (q_1, q_2, g, x_1, x_2, x_{11})$, the set of all possible actions $\Omega(\overrightarrow{S})$ is determined. Let θ be the number of free servers at cluster 1. Thus θ is equal to $\theta = 2 - x_1 - 2x_2 - x_{11}$. Formally,

$$\Omega(\overrightarrow{S}) = \begin{cases} \{0,1\} & \text{, if } g = 2,\ x_{11} = 0,\ \theta \geq 1 \\ \{0\} & \text{, otherwise} \end{cases}$$

Then a specific action is applied. The job at head of queue 1, if any, may enter service or not. If it enters service and another job follows behind in the queue, cases have to be considered with respect to the number of nodes the job that follows requires, so that the resulting states can be determined. Therefore, just after an action d is applied, a set $\Psi_d(\overrightarrow{S})$ is determined. It consists of elements of 2 coordinates: The first one corresponds to a possible state just after an action is applied and the second to the probability by which the state occurs. Below, we give a formal definition of $\Psi_d(\overrightarrow{S})$ for all possible states $\overrightarrow{S}$.

Suppose that action $d = 0$ is applied:

1. $g = 0$: There are no jobs waiting in queue at cluster 1. Therefore, the next state is the same with probability 1. Thus, $\Psi_0(\overrightarrow{S}) = \{(\overrightarrow{S}, 1)\} = \{((q_1, q_2, g, x_1, x_2, x_{11}), 1)\}$.
2. $g = 1$: We distinguish 2 cases:
   - $\theta = 0$: The job does not fit. Again $\Psi_0(\overrightarrow{S}) = \{(\overrightarrow{S}, 1)\}$
   - $\theta \geq 1$: The job fits. We examine whether there exists another job behind or not.
     - $q_1 = 1$: The job enters service immediately and the next immediate state is $(0, q_2, 0, x_1 + 1, x_2, x_{11})$. Therefore, $\Psi_0(\overrightarrow{S}) = \{((0, q_2, 0, x_1 + 1, x_2, x_{11}), 1)\}$
     - $q_1 > 1$: There are 2 possible resulting states:
       * $(q_1 - 1, q_2, 1, x_1 + 1, x_2, x_{11})$, with probability p.
       * $(q_1 - 1, q_2, 2, x_1 + 1, x_2, x_{11})$, probability $1 - p$.

Thus $\Psi_0(\overrightarrow{S}) = \{((q_1-1, q_2, 1, x_1+1, x_2, x_{11}), p), ((q_1 - 1, q_2, 2, x_1 + 1, x_2, x_{11}), 1-p)\}$

3. $g = 2$: This case is analogous to the case $g = 1$, where set $x_2 + 1$ instead of x_2, x_1 instead of $x_1 + 1$, $\theta \leq 1$ instead of $\theta = 0$ and $\theta = 2$ instead of $\theta \geq 1$, since a 2-component job may enter service this time.

Next we assume that action $d = 1$ is taken. Then $\Psi_1(\overrightarrow{S})$ is calculated as follows:

1. $q_1 = 1$: The next state is $(0, q_2, 0, x_1, x_2, x_{11} + 1)$. Therefore, $\Psi_1(\overrightarrow{S}) = \{((0, q_2, 0, x_1, x_2, x_{11} + 1), 1)\}$.

2. $q_1 > 1$: The following cases are possible:

   - $(q_1-1, q_2, 1, x_1, x_2, x_{11}+1)$, with probability p.
   - $(q_1-1, q_2, 2, x_1, x_2, x_{11}+1)$, with probability $1-p$.

   Therefore, $\Psi_1(\overrightarrow{S}) = \{((q_1 - 1, q_2, 1, x_1, x_2, x_{11} + 1), p), ((q_1 - 1, q_2, 2, x_1, x_2, x_{11} + 1), 1 - p)\}$.

After an action is applied the state changes instantaneously. The process stays in the new state for an exponentially distributed amount of time. For every $(\overrightarrow{X}, q) \in \Psi_d(\overrightarrow{S})$, where $\overrightarrow{X} = (q_1', q_2', g', x_1', x_2', x_{11}')$, the transition rates $\eta(\overrightarrow{X}, \overrightarrow{S'})$ from state $\overrightarrow{X}$ to a state $\overrightarrow{S'}$ are calculated as follows:

Suppose that the process is in state $\overrightarrow{X}$. The following events may happen:

- An arrival occurs to a non-empty queue, e.g. $q_1' > 0$. Then the next state is $\overrightarrow{S'} = (q_1' + 1, q_2', g', x_1', x_2', x_{11}')$ with rate λ_1.
- An arrival occurs to an empty queue. We have two cases:
  - The arriving job requires 1 server. The next state is $\overrightarrow{S'} = (1, q_2', 1, x_1', x_2', x_{11}')$, with rate $\lambda_1 \cdot p$.
  - The arriving job requires 2 servers. The next state is $(1, q_2, 2, x_1, x_2, x_{11} + 1)$, with rate $\lambda_1 \cdot (1 - p)$.
- A job arrives at queue 2. The next state is $\overrightarrow{S'} = (q_1', q_2' + 1, g', x_1', x_2', x_{11}')$ at rate λ_2.
- A 1-component job finishes execution from cluster 1. It can only happen when $x_1' > 0$. The resulting state is $\overrightarrow{S'} = (q_1', q_2', g', x_1' - 1, x_2', x_{11}')$ at rate $\mu_1 \cdot x_1'$.
- A 2-component job that is executed locally finishes execution at cluster 1. This is possible for states such that $x_2' > 0$. The next state is $\overrightarrow{S'} = (q_1', q_2', g', x_1', x_2' - 1, x_{11}')$ with rate $\mu_1 \cdot x_2'$.
- A co-allocated job finishes execution. The condition $x_{11}' > 0$ should hold. At rate $\mu_3 \cdot x_{11}'$ $\overrightarrow{S'} = (q_1', q_2', g', x_1', x_2', x_{11}' - 1)$.
- A job at cluster 2 finishes execution. This can only happen when $x_{11}' = 0$, $q_2' > 0$. The next state is $\overrightarrow{S'} = (q_1', q_2' - 1, g', x_1', x_2', x_{11}')$ at rate μ_2.
- No other event can happen.

Therefore, it follows that $\eta(\overrightarrow{X}, \overrightarrow{S'}) =$

- λ_1, if $\overrightarrow{S'} = (q_1' + 1, q_2', g', x_1', x_2', x_{11}')$, $q_1' > 0$
- $\lambda_1 \cdot p$, if $\overrightarrow{S'} = (1, q_2', 1, x_1', x_2', x_{11}')$, $q_1' = 0$
- $\lambda_1 \cdot (1-p)$, if $\overrightarrow{S'} = (1, q_2', 2, x_1', x_2', x_{11}')$, $q_1' = 0$
- λ_2, if $\overrightarrow{S'} = (q_1', q_2' + 1, g', x_1', x_2', x_{11}')$
- $\mu_1 \cdot x_1'$, if $\overrightarrow{S'} = (q_1', q_2', g', x_1' - 1, x_2', x_{11}')$, $x_1' > 0$
- $\mu_1 \cdot x_2'$, if $\overrightarrow{S'} = (q_1', q_2', g', x_1', x_2' - 1, x_{11}')$, $x_2' > 0$
- $\mu_3 \cdot x_{11}'$, if $\overrightarrow{S'} = (q_1', q_2', g', x_1', x_2', x_{11}' - 1)$, $x_{11}' > 0$
- μ_2, if $\overrightarrow{S'} = (q_1', q_2' - 1, g', x_1', x_2', x_{11}')$, $x_{11}' = 0$, $q_2' > 0$
- 0, otherwise.

We need to pay attention when an arrival occurs to empty queue at cluster 1 (case $q_1' = 0$). The next state depends on the number of nodes required by the arriving job, and therefore, 2 cases have to be considered.

The total rate out of state $\overrightarrow{S}$ if action $d \in \{0, 1\}$ is applied, is given by

$$r_d(\overrightarrow{S}) = \sum_{\{(\overrightarrow{X}, q) \in \Psi_d(\overrightarrow{S})\}} q \sum_{\{\overrightarrow{Y} : \eta(\overrightarrow{X}, \overrightarrow{Y}) > 0\}} \eta(\overrightarrow{X}, \overrightarrow{Y})$$

3 THE OPTIMAL POLICY

The model introduced in the previous section can be formulated as a continuous-time Markov decision process. This is our intention, since it leads to the computation of the optimal "co-allocation" policy. We describe the steps we follow so that our model complies with the existing theory of Markov decision processes. A brief overview of the general theory for continuous-time Markov decision processes with the discounted cost as an optimality criterion is given. For a more detailed description, the interested reader should refer to [Bertsekas, 1987].

We consider a continuous-time Markov process with discrete state space. At each state i, an action u is applied. Then the process stays at state i for an exponentially distributed amount of time with parameter $\nu_i(u)$ and switches to another state j with probability $p_{i,j}(u)$. Furthermore, a cost of $g(i,u)$ per unit of time is incurred during the stay of the process at state i. The interest is in minimizing the expected total discounted cost incurred by the system under the set of all possible policies. More formally, we seek to minimize the quantity,

$$E\left\{\int_0^\infty e^{-\beta t} g[x(t),u(t)]dt\right\},$$

where $x(t), u(t)$ are the state and action at time instant t. It is assumed that x, u remain constant between transitions. In addition, β is a real positive number that is called the *Discount Factor*.

An important assumption of the model is that the process has bounded transition rates. More formaly, there exists ν, such that,

$$\nu_i(u) \leq \nu,$$

for all i, u.

Under the previous assumptions, it can be proved (see [Bertsekas, 1987], p.275-280) that the continuous-time decision problem can be converted into a discrete-time one. Specifically, a discrete-time Markov decision chain is defined with transition probabilities,

$$\widetilde{p}_{i,j}(u) = \begin{cases} \frac{\nu_i(u)}{\nu} p_{i,j}(u), i \neq j \\ \\ 1 - \frac{\nu_i(u)}{\nu}, \ i = j, \end{cases}$$

and cost per stage

$$\widetilde{g}(i,u) = \frac{1}{\beta+\nu} g(i,u),$$

for all i, u. This discrete Markov decision problem is proved equivalent to the continuous one. In that proof, a technique called *Uniformization Technique* is applied, which is only applicable to continuous-time Markov processes with bounded transition rates.

One important recursive equation that holds for Markov decision processes with bounded costs is *"Bellman's equation"* (see [Bertsekas, 1987]). Bellman's equation is important for the computation of optimal policies. For the above discrete-time Markov chain it takes the following form:

$$J(i) = \frac{1}{\beta+\nu} min_{u \in U(i)} \{g(i,u) + (\nu - \nu_i(u))J(u) + \nu_i(u) \sum_j p_{i,j}(u) J(j)\},$$

where $U(i)$ is the set of all possible actions that can be applied in state i and $J(i)$ is the optimal mean total discounted cost incurred by an optimal policy given that i is the initial state. In other words, function J is the optimal cost function.

A computational procedure called *Successive Approximation Method* is applied for the computation of optimal policies. The Successive Approximation method is only applicable to Markov decision processes with finite state space. The method starts with an initial guess of J and converges to the optimal cost function after a finite number of iterations. At each iteration, Bellman's recursive equation is used to compute the next value of the optimal cost function. Upon the last iteration, the algorithm yields an optimal *Stationary* policy. A policy is called *stationary*, if it depends on the current state only and not on time. In fact, a stationary policy is a function from the state space to the set of all possible actions.

Our specific model is a continuous-time Markov decision problem in discrete state space. Furthermore, all transition rates are bounded. We have that,

$$\nu_{\overrightarrow{S}}(d) \leq \lambda_1 + \lambda_2 + 2\mu_1 + \mu_2 + \mu_3$$

for all states $\overrightarrow{S}$ and actions $d \in U(\overrightarrow{S})$

Therefore, our model complies with the above theory and can be converted to an equivalent discrete-time Markov decision problem.

However, the state space is infinite. We truncate it, so that we can make the computation

of the optimal stationary policy tractable. We define an integer $\Delta > 0$, such that $q_1, q_2 \leq \Delta$. By bounding the queue lengths in both queues we make our state space finite. Furthermore, the cost per state becomes bounded. Then, the Successive Approximation algorithm can be applied. We compute the optimal policy for the truncated version of the state space. Bellman's equation becomes:

$$J(\overrightarrow{S}) = \frac{1}{v+\beta} \cdot min_{d \in U(\overrightarrow{S})} \{c_1(q_1 + x_1 + x_2 + x_{11}) + c_2 q_2 + \\ + \sum_{\{(\overrightarrow{X},q) \in \Psi_d(\overrightarrow{S})\}} q \sum_{\{\overrightarrow{Y}:\eta(\overrightarrow{X},\overrightarrow{Y})>0\}} \eta(\overrightarrow{X},\overrightarrow{Y}) J(\overrightarrow{Y}) + \\ + (v - r_d(\overrightarrow{i}))J(\overrightarrow{S})\}$$

In the section that follows, we discuss in more detail how the Successive Approximation algorithm performs in our case. In addition, the space and time complexity of the algorithm is calculated. Furthermore , the state space truncation makes the computation of the optimal policy approximate. This issue is further discussed in the next two chapters.

4 SPACE AND TIME COMPLEXITY

The state space of the model we introduced consists of $6(\Delta+1)(2\Delta+1)$ states. The Successive Approximation algorithm operates as follows: Initially, a guess of the value of the optimal cost function J is made. These values are stored in a vector matrix with dimension equal to the number of states. At each iteration the algorithm goes through each element in the vector matrix and compares at most two actions. Then it computes a new value for the cost function at the specific state. This new value is stored in another vector matrix. Thus, at each iteration, the previously computed vector matrix is used to compute a new one. This procedure converges to the optimal cost function vector matrix after a finite number of iterations.

At each iteration, we need to do $\boldsymbol{O}(2 \cdot (6(\Delta + 1)(2\Delta + 1)))$ checks in total. As a result, the total number of checks our algorithm needs is $\boldsymbol{O}(6(\Delta + 1)(2\Delta + 1)2n)$, where n is the total number of iterations.

Obviously the time complexity of the algorithm increases when the truncation level Δ increases. Furthermore, we need to store two vector matrices of dimension equal to the state space. Therefore, the space complexity of the algorithm increases with Δ.

The state space truncation may result in errors when computing the optimal policy. The optimal policy for the truncated version of the state space may be different from the optimal policy for our initial infinite state space problem. The greater the truncation level Δ is, the more probable is that the policy we compute is indeed the optimal one for the initial problem. Thus, we need to keep a balance between the accuracy of our results and the time and space complexity of our model. We choose $\Delta = 30$ which yields 11346 states. We discuss this issue in the next chapter as well.

The rate of convergence of the successive approximation algorithm depends on the discount factor β. The closer β is to zero, the greater is the number of iterations the algorithm needs to converge. We set $\beta = 0.05$ in our experiments. We choose a value close to zero since we are interested in the behavior of the system when operating for a long time. However, this comes at a time cost. With $\Delta = 30$ and $\beta = 0.05$, the time needed for each run was nearly 5 hours. Furthermore, the algorithm needed approximately 1100 iterations to converge to the optimal cost function vector.

5 EXPERIMENTAL RESULTS

In this section we present two specific cases for which the optimal "co-allocation" policy is computed. We define as m the mean number of servers required by jobs submitted to cluster 1. Therefore, $m = p + 2(1-p)$. The load at cluster 1 is denoted as ρ_1 and is defined as,

$$\rho_1 := \frac{\lambda_1 m}{2\mu_1}.$$

The load at cluster 2 is given by,

$$\rho_2 := \frac{\lambda_2}{\mu_2}.$$

The total load in the system is

$$\rho := \frac{\rho_1 + \rho_2}{2}.$$

Co-allocation is only possible for states $\overrightarrow{S} = (q_1, q_2, g, x_1, x_2, x_{11})$, such that $x_{11} = x_2 = 0$, $g = 2$, $q_1 > 0$, $0 \leq x_1 \leq 1$.

In the first experiment we choose $p = 0.6$, $\mu_1 = \mu_2 = 1$. Furthermore, we set $\rho_1 = \rho_2 = \rho = 0.8$.

Thus, the system operates in heavy load conditions.

As stated in the previous section, in both experiments we set $\Delta = 30$. We choose such a value for Δ since our purpose is to compute a policy that is close to the optimal policy for the initial infinite queue length problem. We need to note that the optimal policy computed in this section is an approximation of the theoretical optimal policy. This is because we cannot be sure about how the theoretical optimal policy behaves for values $q_1, q_2 > \Delta$.

The service rate for co-allocated jobs is set to $\mu_3 = 0.7 < \mu_1,\ \mu_2$. Service times for co-allocated jobs are greater than those of local jobs because of the additional delays that the slow inter-cluster communication network imposes.

Our results show that the optimal policy when $x_1 = 0$ is "do not co-allocate job". Therefore, co-allocation when both servers in cluster 1 are free seems not a good choice. The optimal policy for states such that $x_1 = 1$ is depicted in figure 2. It remains the same for states $16 \leq q_1, q_2 \leq 30$.

$x_1 = 1$	q_2																
q_1		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	3	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	4	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	5	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	6	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	7	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	8	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	9	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
	10	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
	11	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
	12	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
	13	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
	14	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
	15	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0

Figure 2: Optimal policy for $\Delta = 30$, $\mu_1 = \mu_2 = 1, \mu_3 = 0.7, p = 0.6, \rho_1 = \rho_2 = 0.8, \beta = 0.05, c_1 = 2,\ c_2 = 1,\ x_1 = 1$.

We observe that co-allocating jobs when there is exactly one free server at cluster 1 is beneficial. However, the optimal co-allocation policy depends on the relative magnitude of the number of customers waiting in each of the two queues. The fact that co-allocation is beneficial in the case $x_1 = 1$ only, indicates that trying to better "pack" incoming jobs in the system, improves performance.

In the second experiment we increase the load in both clusters, so that the system operates under overload conditions. We set $\rho = \rho_1 = \rho_2 = 0.9$. Additionally, we choose $\mu_3 = 0.8$, $p = 0.8$. This means that co-allocated jobs are executed faster than in the first experiment. The rest of the parameters are kept the same as in the previous experiment. The optimal policy for states $x_1 = 0$ is depicted in figure 3. It remains the same for values $6 < q_1,\ q_2 \leq 30$.

$x_1 = 0$	q_2							
q_1		0	1	2	3	4	5	6
	1	0	0	0	0	0	0	0
	2	0	0	0	0	0	0	0
	3	0	0	0	0	0	0	0
	4	1	0	0	0	0	0	0
	5	1	0	0	0	0	0	0
	6	1	0	0	0	0	0	0

Figure 3: Optimal policy for $\Delta = 30$, $\mu_1 = \mu_2 = 1, \mu_3 = 0.8, p = 0.8, \rho_1 = \rho_2 = 0.9, \beta = 0.05, c_1 = 2,\ c_2 = 1,\ x_1 = 0$.

Figure 4 illustrates the optimal policy when $x_1 = 1$. It does not change for $6 < q_1 \leq \Delta,\ 10 < q_2 \leq \Delta$.

$x_1 = 1$	q_2											
q_1		0	1	2	3	4	5	6	7	8	9	10
	1	1	0	0	0	0	0	0	0	0	0	0
	2	1	1	1	0	0	0	0	0	0	0	0
	3	1	1	1	1	1	1	0	0	0	0	0
	4	1	1	1	1	1	1	1	1	1	1	0
	5	1	1	1	1	1	1	1	1	1	1	0
	6	1	1	1	1	1	1	1	1	1	1	0

Figure 4: Optimal policy for $\Delta = 30$, $\mu_1 = \mu_2 = 1$, $\mu_3 = 0.8$, $p = 0.8$, $\rho_1 = \rho_2 = 0.9$, $\beta = 0.05$, $c_1 = 2$, $c_2 =$, $x_1 = 1$.

Again we observe that even in that heavy load conditions, co-allocation when both servers at cluster 1 are free is only beneficial when $q_2 = 0, q_1 > 3$. On the other hand, it is fully employed when $x_1 = 1$. This indicates that co-allocation under high load conditions becomes more beneficial, a result stated in [Bucur and Epema, 2003] as well. Furthermore, co-allocating parallel jobs is more advantageous as the service rate of co-allocated jobs increases.

6 ANALYSIS BY SIMULATION

We compare the optimal policy as computed in the previous section with two other policies. Simulation is used so that policies are compared with respect to the average cost criterion. This is a more suitable measure for our case since we are interested in the long run behavior of the system. In particular, the estimate of $E[c_1(q_1 + x_1 + x_2 + x_{11}) + c_2 q_2]$ is computed by simulation. 2.000.000 jobs completions were simulated and a 95% confidence interval was com-

puted by 60 independent replications.

Many policies were checked and many experiments were conducted. We introduce a heuristic policy which showed better performance. Additionally, a policy that does not employ co-allocation at all is used for comparison. Specifically, we define the following policies:

1. The *"Local"* policy. This policy does not employ co-allocation at all. All jobs are executed locally to the cluster they were initially submitted. It is used for comparison with dynamic co-allocation policies.

2. The *"$pc_1c_2 > c_2p_2$-Best pack"* policy. This is a heuristic policy that co-allocates the job at head of queue 1 if the conditions $p \cdot c_1q_1 > c_2q_2$, $x_1 = 1$ hold. It attempts to best "pack" jobs in the system, while co-allocating jobs when the cost in cluster 1 becomes greater than cost in cluster 2. The multiplier p in the first part of the inequality is used to prevent the scheduler from doing too many co-allocations when there are many 2-component jobs in the system. By intuition, the greater the percentage of single component jobs is, the more probable is that a 1-component job follows behind the co-allocated one. As a result the 1-component job starts execution in the next scheduling decision. If a 2-component job follows, it is then blocked and has to wait for the co-allocated job to finish execution. Thus jobs experience greater response times and utilization of the system drops. We find by experiment that this policy outperforms the policy that co-allocates jobs when $c_1q_1 > c_2q_2$ and $x_1 = 1$ hold. Furthermore, as stated in the previous section, co-allocations when conditions $x_1 = 0$ is not recommended. That is why a "best pack" policy is preferred.

In all our experiments we assume that $\mu_1 = \mu_2 = 1$. Additionally, we set $\rho = \rho_1 = \rho_2$.

In the first simulation experiment, we keep $\mu_3 = 0.7$. Our purpose is to examine how our heuristic policy performs as the total load in the system increases. The parameters values for the simulation as well as our results are illustrated in figure 5. Our first observation is that our policy performs too close to optimal under moderate load conditions. However, at loads > 0.85, heuristic policy fails to respond as good as optimal policy. In addition, it performs better that local policy in all cases. We have experimented with more elaborate policies but none of them could respond better than the heuristic we propose under overload conditions.
If we compare the optimal policy with just the local one, we can plot the % reduction caused by the optimal policy. The graph is depicted in figure 6. We find that optimal policy causes significant reduction in average cost payed. This reduction increases as the total load in the system increases. As a result, co-allocation is more advantageous under high load conditions. This is a result stated in [Bucur and Epema, 2003] as well. Furthermore, there may exist policies, even better than ours, that cause significant improvement in system performance.

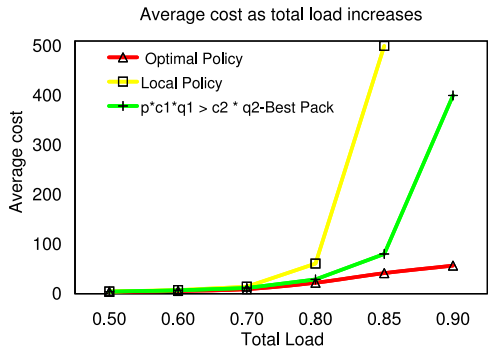

Figure 5: Simulation results for $\mu_1 = \mu_2 = 1$, $\mu_3 = 0.7$, $p = 0.6$, $\rho_1 = \rho_2 = 0.5 - 0.9$, $c_1 = 2$, $c_2 = 1$.

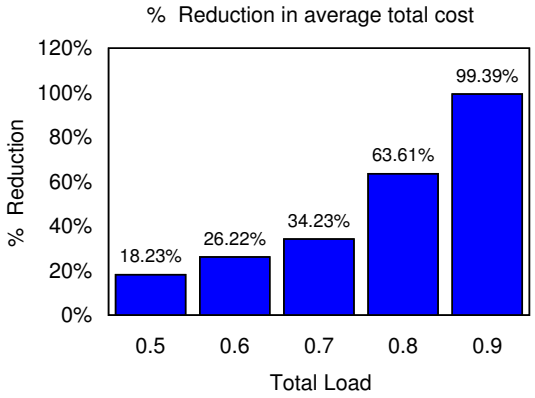

Figure 6: % Reduction caused by optimal policy in cost incurred by local one. Parameter values: $\mu_1 = \mu_2 = 1$, $\mu_3 = 0.7$, $p = 0.6$, $\rho_1 = \rho_2 = 0.5 - 0.9$, $c_1 = 2$, $c_2 = 1$.

In the second experiment, we examine how the choice of costs affects the performance of the heuristic policy. In this case, we assume that costs at both queues are equal. This means that all jobs submitted in the system are of equal importance. Service rate for co-allocated jobs is kept fixed at 0.7. The total load in the system is increased until saturation is reached. Our results

are depicted in graph 7.

Heuristic policy performs to close to optimal even in high load conditions. As a general result, our heuristic policy performs too close to optimal when the system does not operate under overload conditions. Additionally, it is not affected by the choice of cost in each cluster.

So far, we have selected $\mu_3 = 0.7$. How does the service rate for co-allocated jobs affects the performance of our heuristic policy? To answer this question we are conducting another experiment. In this experiment we set the load fixed at $\rho = \rho_1 = \rho_2 = 0.8$. The service rate for co-allocated jobs varies between 0.4 and 0.8. Our results are indicated in graph 8.

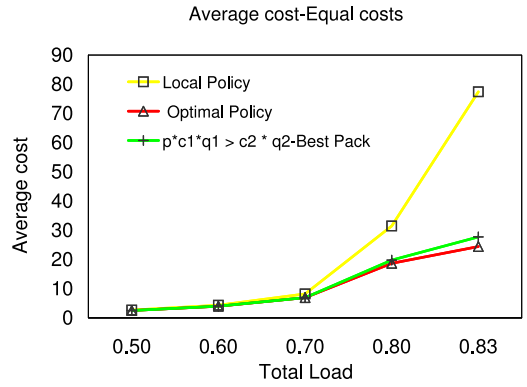

Figure 7: Simulation results for $\mu_1 = \mu_2 = 1$, $\mu_3 = 0.7$, $p = 0.6$, $\rho_1 = \rho_2 = 0.5 - 0.83$, $c_1 = c_2 = 1$.

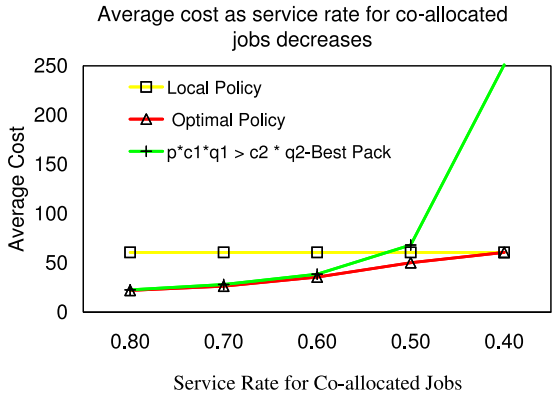

Figure 8: Simulation results for $\mu_1 = \mu_2 = 1$, $\mu_3 = 0.4 - 0.8$, $p = 0.6$, $\rho_1 = \rho_2 = 0.8$, $c_1 = 2, c_2 = 1$.

It is obvious that co-allocation is beneficial for $\mu_3 \geq 0.4$. For $\mu_3 < 0.4$, co-allocating jobs is not beneficial since the optimal policy is the same as the local one. Our heuristic policy performs very well when $\mu_3 \geq 0.5$. In fact, for service rates $\mu_3 \geq 0.6$, it is too close to optimal one. However, it fails to detect the fact at service rates $\mu_3 < 0.4$, jobs need not be co-allocated. We experimented with a number of conditions that may detect this fact, but non of them could perform well in all our experiments. As a conclusion, there is a threshold value θ, such that for $\mu_3 < \theta$, co-allocating parallel jobs is not at all beneficial. The value of $\theta = 0.4$ seems to be of significant importance since our optimal policy becomes the same as the local one for $\mu_3 \leq 0.4$ in all our experiments.

In the last experiment, we examine the impact of the percentage of single components jobs, e.g. p, on the performance of our heuristic policy. Four values of p are examined and our results are depicted in figure 9.

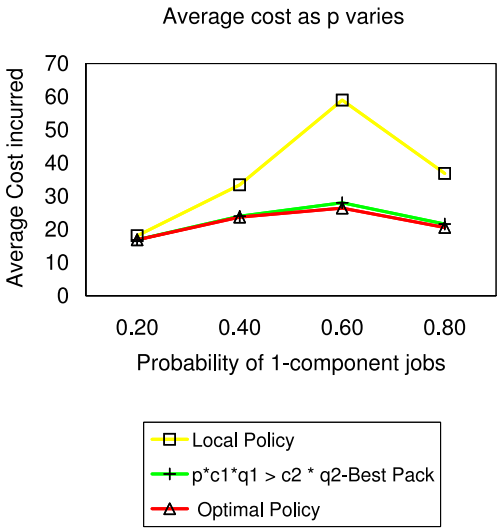

Figure 9: Simulation results for $\mu_1 = \mu_2 = 1$, $\mu_3 = 0.7$, $p = 0.2 - 0.8$ $p = 0.6$, $\rho_1 = \rho_2 = 0.8$, $c_1 = 2, c_2 = 1$.

Firstly, we observe that heuristic policy performs extremely well since it is too close to optimal one. Therefore, it is a robust policy concerning variability in the percentage of 1-component jobs in the workload. In addition, there is an increase in average cost incurred as the percentages of two kind of jobs becomes more "balanced". Therefore, systems with workloads with many jobs of one type and a few jobs from another type seem perform better. An explanation for that is that jobs are packed better in the system and utilization of servers increases in that way. As a conclusion, parameter p plays an important role in the performance of the system and has to be taken in consideration of all heuristic policies.

7 GENERAL RESULTS

We briefly state our results from analysis in the previous sections:

1. Co-allocating jobs among different clusters is beneficial if it is done dynamically. Co-allocation becomes more beneficial as system load increases.
2. Policies that "pack" better jobs in the system outperform policies that do not take that into account. However, if inter-cluster communication speed is high enough, co-allocation may be beneficial even when jobs are not packed well in the system
3. There is a threshold value with respect to the service rate of co-allocated jobs such that co-allocating jobs with service rates less than that value is not at all beneficial. We find that a lower bound for co-allocating jobs is when inter-cluster communication speed is the 40% of intra-cluster communication speed.
4. A balance between single and 2-component jobs in the workload seems to decrease performance. However, the negative effect of job size variability in average performance can be overcome if dynamic co-allocation policies are applied.

8 CONCLUSION

A Dynamic Programming model was introduced to investigate the possibility of dynamically co-allocating jobs in multi-cluster systems. Since complex models are in general intractable, a simple model is introduced to examine the performance under co-allocation. The optimal policy for the truncated state space is computed. Simulation is used to compare the optimal policy with the local policy and a heuristic one. Our results indicate there is significant room for policies that perform much better than the local policy even when co-allocated jobs receive service at much lower rate than other jobs. Furthermore, a heuristic policy was proposed that performs close to optimal for a wide range of parameters. We note that in case a co-allocation policy is independent of the queue length of cluster 2, the problem can be studied by static analysis. Although such a policy may be suboptimal, it is more likely to be efficiently implemented in a distributed protocol because only local information is required. This may be the subject of future work. One can extend the model by adding more job types, more clusters, and more nodes in each cluster. However the resulting problem is a complex one to solve.

References

Bertsekas D. P., *"Dynamic Programming: Deterministic and Stochastic Models"*, 1987, Prentice-Hall.

Bucur A.I.D. and Epema D.H.J, *"The performance of Processor Co-allocation in multicluster systems"*, 3rd IEEE/ACM Int'l Symp. on Cluster Computing and the Grid (CCGrid2003), Tokyo, Japan, may 2003, 302-309, 2003.

Ernemann C., Hamscher V., Schwiegelshohn U.,Yahyapour R. and Streit A., *"On Advantages of Grid Computing for Parallel Job Schedulling"*, Cluster Computing and the Grid, 2nd IEEE/ACM International Symposium CCGRID 2002, pages 39 - 46, 2002.

Karatza H.D., *"Task Scheduling Performance in Distributed Systems with Time Varying Workload"*, Neural, Parallel & Scientific Computations, Dynamic Publishers, Atlanta, 10, 2002, pp. 325-338.

Law A. M. and Kelton W. D. , *"Simulation Modeling and Analysis"*, Mc Graw Hill, 2000, Third Edition.

Martin S. and Mitrani I., *"Dynamic Routing Between Two Queues with Unreliable Servers"*, Int. Journal of SIMULATION: Systems, Science & Technology, UK Simulation Society (to appear).

Palmer J. and Mitrani I., *"Dynamic Server Allocation in Heterogeneous Clusters"*, in Proceedings of the First International Working Conference on Performance Modelling and Evaluation of Heterogeneous Networks (HETNETs 2003) Kouvatsos, D. pp. 12/1-12/10 2003.

Palmer J. and Mitrani I., *"Optimal Server Allocation in Reconfigurable Clusters with Multiple Job Types"*, in Proceedings of the Computational Science and its Applications (ICCSA 2004). International Conference, Assisi, Italy, May 14-17, 2004. Pt. 2 Lagana, A., Gavrilova, M.L. , Kumar, V. et al (eds) Lecture Notes in Computer Science Volume 3044 pp. 76-86 Springer 2004.

Riska A., Sun W., Smirni E., Ciardo G., *"ADAPTLOAD: effective balancing in clustered web servers under transient load conditions"*, in Proceedings of the 22nd International Conference on. Distributed Computing Systems, (ICDCS 2002), Vienna, Austria, July 2002, pp. 104-111.

Biographies

Dimitrios Filippopoulos received in 2001 his BSc degree in Mathematics from Aristotle University of Thessaloniki, Greece and in 2002 the MSc Degree in Advanced Computing from Imperial College London, UK. He is currently a PhD Student at the Department of Informatics, Aristotle University of Thessaloniki, Greece, in the area of Performance Modelling and Evaluation of Computer Systems. His research interests include queueing models of parallel and distributed systems. His email and web address are <fildim@csd.auth.gr>, and <agent.csd.auth.gr/∼ fildim>.

Helen Karatza is an Associate Professor in the Department of Informatics at the Aristotle University of Thessaloniki, Greece. Her research interests mainly include Performance Evaluation of Parallel and Distributed Systems, Multiprocessor Scheduling, Mobile Agents, Mobile Computing, and Simulation. Dr. Karatza is a member of the Editorial Board of the International Journal of Simulation: Systems, Science & Technology (the UK Simulation Society), Associate Editor of the Journal Simulation: Transactions of the Society for Modeling and Simulation International (Applications Section), and area Editor for computer systems of the Journal of Systems and Software (Elsevier). She has served as a member of Program Committees and Program Chair of many Simulation related International Conferences/Symposia. Her email and web address are <karatza@csd.auth.gr>, and <agent.csd.auth.gr/∼ karatza>.