2010-10-28

Packing Virtual Machines onto Servers

David Luke Wilcox
Brigham Young University - Provo

Packing Virtual Machines onto Servers

David Wilcox

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Kevin Seppi, Chair
Kelly Flanagan
Sean Warnick

Department of Computer Science

Brigham Young University

December 2010

ABSTRACT

Packing Virtual Machines onto Servers

David Wilcox

Department of Computer Science

Master of Science

Data centers consume a significant amount of energy. This problem is aggravated by the fact that most servers and desktops are underutilized when powered on, and still consume a majority of the energy of a fully utilized computer even when idle This problem would be much worse were it not for the growing use of virtual machines. Virtual machines allow system administrators to more fully utilize hardware capabilities by putting more than one virtual system on the same physical server.

Many times, virtual machines are placed onto physical servers inefficiently. To address this inefficiency, I developed a new family of packing algorithms. This family of algorithms is meant to solve the problem of packing virtual machines onto a cluster of physical servers. This problem is different than the conventional bin packing problem in two ways. First, each server has multiple resources that can be consumed. Second, loads on virtual machines are probabilistic and not completely known to the packing algorithm.

We first compare our developed algorithm with other bin packing algorithms and show that it performs better than state-of-the-art genetic algorithms in literature. We then show the general feasibility of our algorithm in packing real virtual machines on physical servers.

Contents

Contents iii

List of Figures iv

List of Tables v

1 Introduction 1

1.1 Related Work . . . 2

1.1.1 Bin Packing . . . 3

1.1.2 Genetic Algorithms . . . 5

1.1.3 Load Rebalancing . . . 6

1.2 Thesis Statement . . . 7

1.3 Thesis Format . . . 7

2 Solving the Multi-Resource Bin Packing Problem with a Reordering Grouping Genetic Algorithm 9

2.1 Introduction . . . 10

2.2 Bin Packing . . . 11

2.2.1 Multi-Capacity Bin Packing . . . 13

2.2.2 Solutions to the Bin Packing Problem . . . 13

2.3 Reordering Grouping Genetic Algorithm . . . 16

2.3.1 Algorithm Overview . . . 17

2.3.2 Fitness Function . . . 18

2.3.3 Crossover Operator . . . 20

2.3.4 Mutation Operator 21
2.4 Experimental Setup 22
2.4.1 Conventional Bin Packing Problem 22
2.4.2 Multi-Capacity Bin Packing Problem 23
2.4.3 RGGA For Virtual Machine Deployments 23
2.5 Results 24
2.5.1 Conventional Bin Packing Problem 24
2.5.2 Multi-Capacity Bin Packing Problem 26
2.5.3 RGGA For Virtual Machine Deployments 30
2.6 Conclusions 33

3 Probabilistic Virtual Machine Assignment 34
3.1 Introduction 35
3.2 Background Research 36
3.2.1 Virtual Machine Assignment 37
3.2.2 Conventional Bin Packing 37
3.2.3 Multi-Capacity Bin Packing Problem 39
3.2.4 Genetic Algorithms 41
3.3 Description of Model 42
3.3.1 Probabilistic Estimates 42
3.4 Initial Assignment Algorithms 44
3.4.1 Worst Fit 45
3.4.2 First Fit 45
3.4.3 Permutation Pack 46
3.4.4 Reordering Grouping Genetic Algorithm 46
3.5 Metrics to Determine Success 47
3.5.1 Number of Servers Used 47
3.5.2 Proportion of Server Resources Over Capacity 47

3.6 Experimental Setup 48
3.7 Results 50
3.7.1 Repacking to Servers 52
3.8 Conclusion 53

4 An Analysis of Variance in Virtual Machine Packing 55
4.1 Introduction 56
4.2 Experimental Setup 57
4.3 Results 58
4.3.1 Virtual Machines Have Equal Variance 58
4.3.2 Virtual Machines Have Differing Variance 62
4.4 Conclusion 65
4.5 Future Work 65

5 Virtual Machine Migration 66
5.1 Introduction 67
5.2 Load Redistribution Algorithm 68
5.3 Experimental Setup 68
5.4 Results 70
5.5 Conclusion 75

6 Conclusions and Future Work 77
6.1 Conclusion 77
6.1.1 Variants of RGGA 78
6.2 Future Work 80

References 82

List of Figures

1.1 One way to visualize a virtual machines in a server. 3

1.2 An inefficient virtual machine packing solution. 4

1.3 An efficient virtual machine packing solution. 5

2.1 The two-capacity bin packing problem. The items in each bin are packed regardless of order. Resources used by each item are additive in each dimension. 12

2.2 The conventional two-dimensional bin packing problem. The position and orientation of items in a bin matter. 12

2.3 The average number of function evaluations for the exon shuffling GA are compared with the function evaluations for RGGA.[1] 25

2.4 The average improved solutions for each mutation type for the multi-capacity problems. 28

2.5 The average percent of mutations that improved individuals plotted against the number of items in each of MCBPPs. 29

2.6 The average percent of mutations that improved individuals for single capacity problems. 29

2.7 The amount of energy used in solutions returned by the three different algorithms. 30

2.8 The average number of bins in the best solution for different iteration numbers for HARD0. 31

2.9 The amount of time taken to find a solution for varying sizes of Multi-Capacity Bin Packing Problems. 32

2.10 The number of servers found by solutions of RGGA, First Fit Decreasing (FFD) and Permutation Pack (PP). 33

3.1 An inefficient Bin Packing solution. 38
3.2 An efficient Bin Packing solution. 38
3.3 An inefficient Multi-Capacity Bin Packing solution. 40
3.4 An efficient Multi-Capacity Bin Packing solution. 40
3.5 The pdf, cdf and icdf for the normal distribution with mean $\mu = 0$ and variance $\sigma^2 = 1$. 43
3.6 A comparison of the proportion of servers over capacity for varying percentile values. 51
3.7 A comparison of the the number of servers found for varying percentile values. 51
3.8 A comparison of the proportion of servers over capacity with the number of servers found. 52
3.9 A comparison of the proportion of solutions over capacity that our algorithm predicted and also the measured proportion of solutions over capacity when deploying real VMs to servers for the 80th cdf percentile. 53

4.1 The number of servers found for varying algorithms for varying percents of servers over capacity for a problem with very low variance. 59
4.2 The number of servers found for varying algorithms for varying percents of servers over capacity for a problem with high variance. 60
4.3 The number of servers found for varying algorithms vs percent of servers over capacity. Variance on VMs is set to 1.7% the total server capacity. 61
4.4 The number of servers found for varying algorithms vs percent of servers over capacity. Variance on VMs is set to 2.5% the total server capacity. 61
4.5 The number of servers found for varying algorithms vs percent of servers over capacity. Variance on VMs is set to 3.1% the total server capacity. 62

4.6 The number of servers found for varying algorithms vs percent of servers over capacity. Half of VMs have high variance. Half of VMs have low variance. . . 63
4.7 The number of servers found for RGGA and RGGA w/ Part vs percent of servers over capacity. Half of VMs have high variance. Half of VMs have low variance. 64

5.1 A surface graph showing the number of servers yielded by running RGGA at different icdf values and different targets with low variance. 71
5.2 A surface graph showing the number of servers yielded by running RGGA at different inverse cumulative distribution values and different targets with average variance. 72
5.3 A surface graph showing the number of servers yielded by running RGGA at different icdf values and different targets with high variance. 73
5.4 A surface graph showing the number of servers yielded by running RGGA at different icdf values and different values of target percent of servers over capacity with low variance. 74
5.5 A surface graph showing the number of servers yielded by running RGGA at different icdf values and different values of target percent of servers over capacity with medium variance. 74
5.6 A surface graph showing the number of servers yielded by running RGGA at different icdf values and different values of target percent of servers over capacity with high variance. 75

6.1 The number of servers found for varying algorithms vs percent of servers over capacity. Half of VMs have high variance. Half of VMs have low variance. . . 80

List of Tables

2.1 Our algorithm compared with other algorithms on the same 10 hard conventional bin packing problem instances. These instances are not representative of MCBPP. 24

2.2 Summary of RGGA compared with Rolfshagen et al.'s Exon Shuffling approach.[2] 25

2.3 The average, minimum and maximum number of function evaluations for RGGA on the single-capacity data set used. 26

2.4 The average, minimum and maximum function evaluations for exon shuffling genetic algorithm on the single-capacity data set used. 26

2.5 Summary of our experimental results of our algorithm on the multi-capacity problems. 27

Chapter 1

Introduction

Servers and desktops use an enourmous amount of energy. In 2005, data centers in the United States accounted for 1.2% of all U.S. energy consumption spending $2.7 billion [12]. One major problem exacerbating this is that most servers are in use only 5-15% of the time they are powered on, yet most x86-based hardware consumes 60-90% of the energy required by a fully utilized server even when idle [4].

Data center costs can be reduced by utilizing virtual machines. Using virtualization, system administrators can put multiple virtual machines with associated operating systems on the same physical machine in order to exploit hardware capabilities more fully. Virtualization increases the amount of feasible work that can be done in data centers.

System administrators can consolidate many virtual machines on the same physical machine, saving money on hardware and energy costs. To maximize the savings, administrators should place as many virtual machines onto a server as possible while satisfying certain performance criteria. Using virtual machines, businesses can more fully utilize increasing capabilities of hardware. In order to disambiguate between virtual machines and physical machines, we refer to physical machines as servers.

The problem of packing virtual machines onto servers can be formulated as a special case of the bin packing problem. The bin packing problem is defined as follows: Given a set of items, each with its own weight, find the assignment of items to bins under which the number of bins used is minimized without violating capacity constraints [3].

The problem of packing virtual machines onto servers is not as simple as the conventional bin packing problem. Packing virtual machines is different from the traditional bin packing problem in the following ways:

- In the normal bin packing problem, each item has only one weight and each bin has only one capacity. In the problem of packing virtual machines, each bin has multiple capacities or resources that are independent of one another. Each virtual machine will add an amount of load to the various resources of the server (RAM, CPU, etc.). An algorithm solving this problem must be able to balance loads with respect to all the resources on the server.

- In the normal bin packing problem, weights on items are certain and known ahead of time. However, in the problem of packing virtual machines onto servers, the weights on each item are uncertain. During the packing process, an algorithm solving this problem has only a probabilistic distribution which it uses to model the loads of a particular virtual machine. The future load on each of the resources of the virtual machine can be represented by a probability distribution. This probability distribution can be used both to predict future load as well as sample future load.

There are many different resources available to virtual machines on a server. A virtual machine can slow down or even be brought to a halt if all of any particular resource it needs is already allocated. Therefore, separate load distributions should be developed for each resource on a server.

1.1 Related Work

This section will investigate related work in three parts. First, the bin packing problem, some of its properties, and heuristics for solving it will be discussed. Second, the prior research done in genetic algorithms will be investigated. Third, load balancing will be discussed.

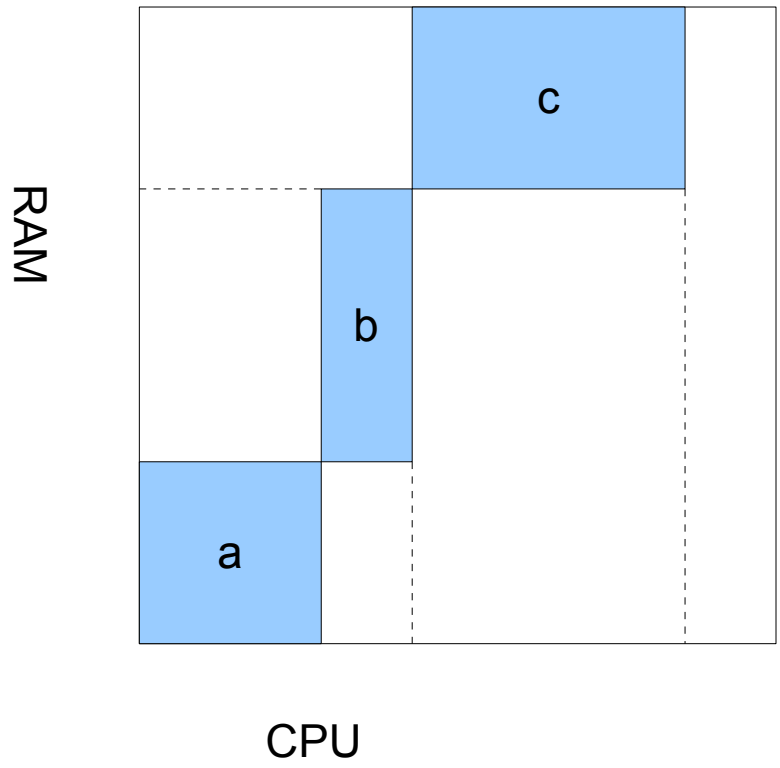

Figure 1.1: One way to visualize a virtual machines in a server.

1.1.1 Bin Packing

The *bin packing problem* is formulated as follows. Given a finite set of n items $I = \{1, 2, \ldots, n\}$ with corresponding weights $W = \{w_1, w_2, \ldots, w_n\}$ and a set of identical bins each with capacity $\mathcal{C}$, find the minimum number of bins into which the items can be placed without exceeding the bin capacity $\mathcal{C}$ of any bin. A solution to the bin packing problem is of the form $B = \{b_1, b_2, \ldots, b_m\}$, where each b_i is the set of items assigned to bin i, and is subject to the following constraints:

1. $\forall i \, \exists! \, j$ such that $i \in b_j$ (Every item has to belong to some unique bin.)

2. $\forall j \, \sum_{n \in b_j} w_n \leq \mathcal{C}$ (The sum of the weights of items inside any bin cannot be greater than the bin capacity.)

However, the bin packing problem discussed here is different from the conventional bin packing problem. One way to visualize it can be seen in Figure 1.1. The vertical dimension of

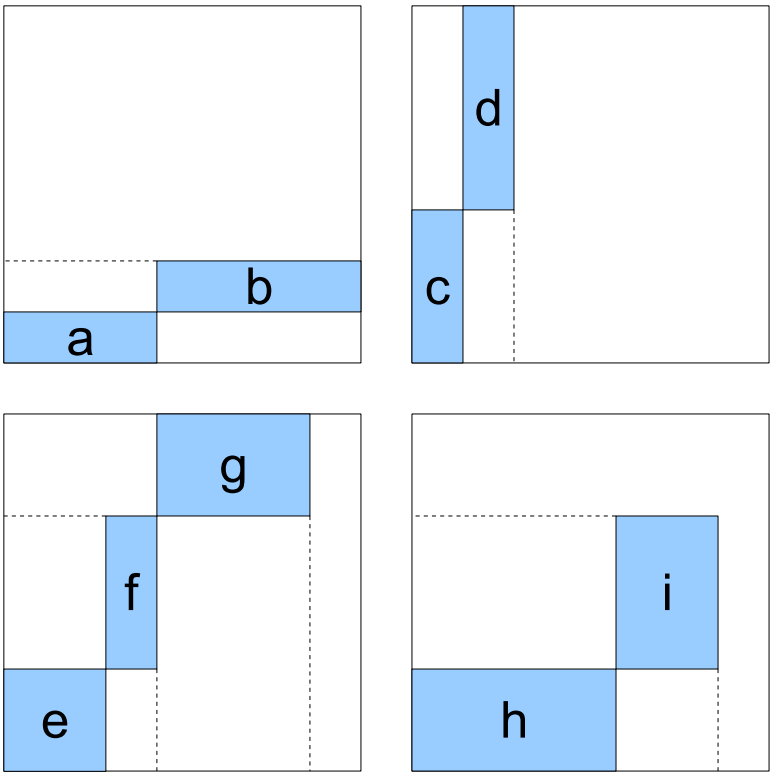

Figure 1.2: An inefficient virtual machine packing solution.

Figure 1.1 refers to the total amount of RAM capacity on a particular server. The horizontal dimension of Figure 1.1 refers to the total amount of CPU capacity on a server. Items are placed on each corner to show that the sum of the weights for any one type has to be less than the corresponding bin capacity. The sum of the CPU taken by all virtual machines must be less than the corresponding CPU capacity of the server. Likewise, the sum of the RAM taken by all virtual machines must be less than the corresponding RAM capacity of the server. This is true if system administrators wish to maintain a reasonable performance criteria, which is an assumption we make here.

As with the conventional Bin Packing Problem, the objective is to minimize the number of bins. Figures 1.2 and 1.3 illustrate this principle. Figure 1.3 packs the exact same items as are found in Figure 1.2 in three bins instead of four. In the Virtual Machine Assignment Problem, this means one fewer server in utilization.

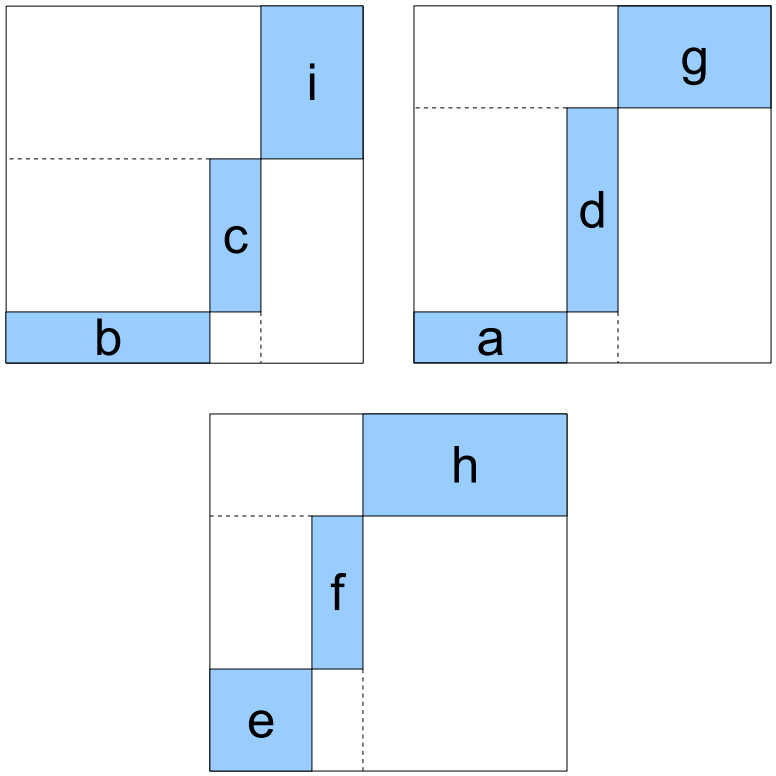

Figure 1.3: An efficient virtual machine packing solution.

1.1.2 Genetic Algorithms

Bin packing problems can also be solved with Genetic Algorithms (GAs). There is no rigorous definition for GAs [15]. GAs derive much of their inspiration from Darwinian biological processes. In GAs, individuals represent candidate solutions to the problem. These candidate solutions explore the solution space by undergoing processes similar to those of biological organisms. The simplest form of a genetic algorithm involves three types of operators:

- **Selection**–Individuals in the population are selected for crossover with other individuals. Usually, selection is based on elitism, where the more fit individuals are selected more often than less fit individuals.

- **Crossover**–Two individuals in the population exchange subsections of their candidate solution with each other to create new offspring.

- **Mutation**–After crossover, each individual has a probability of having their candidate solution modified slightly.

These three operators act on the candidate solutions in the population in order to improve the population [15].

1.1.3 Load Rebalancing

Load rebalancing has traditionally been thought of as, given a suboptimal assignment of jobs to processors, reassign jobs reducing the load on the heaviest processor. This is done by migrating jobs from the heaviest-loaded processor to a less loaded processor [9].

Aggarwal et al.[2] gave a taxonomy of different load rebalancing problems and many proofs about load rebalancing. In general, their theorems show that load rebalancing is an NP Hard problem [2].

Even though the conventional load rebalancing problem is interesting, the problem addressed here is different in two ways. First, our problem does not assume any type of initial assignment of virtual machines to servers, but rather generates an initial assignment. For example, this assignment may be made at the beginning of a business day for an e-commerce web site when loads on virtual machines are low and administrators have more liberty to migrate virtual machines. After the day has started, system administrators may not be able to move around virtual machine as freely because servers need to be dedicated to processing customer transactions, and not performing VM migrations between servers.

The second way that our problem is different is that the optimization criterion of our algorithm is different. The load rebalancing problem minimizes the load on the heaviest server. Instead, we minimize the number of servers in utilization, while assuring that all virtual machines maintain sufficient performance.This difference can also be thought of in the following way. Conventional load rebalancing algorithms swap processes in order to minimize the load on servers with the goal to increase throughput. Instead of measuring throughput, we minimize energy consumption throughout the entire system by turning off servers while

assuring that no server becomes overloaded. Turning off servers is equal to minimizing the number of bins in the bin packing problem. We have found no other algorithms in the literature whose goal is to solve this new variant of load rebalancing.

1.2 Thesis Statement

Genetic algorithms can be adapted to the problem of packing virtual machines tightly onto servers such that unused physical machines can be turned off, saving energy, while assuring that no server becomes overloaded. Primarily, we are interested in answering the following question: "How can the Bin Packing Problem be extended to packing virtual machines onto servers?"

In order to solve this problem, we assert that If VM loads are probabilistic, different estimates from the VM probability distribution can be taken to make VM packing conform to the Bin Packing Problem.

1.3 Thesis Format

We divide the rest of this thesis into four separate papers. In the first paper, we investigate a new type of Bin Packing Problem that can be applied directly to packing virtual machines to servers. Along with this new Bin Packing Problem, we present a genetic algorithm with accompanying operators specifically meant to solve this type of bin packing problem. We show that our new genetic algorithm outperforms state-of-the-art genetic algorithms in literature on the conventional Bin Packing Problem and show the genetic algorithm's applicability in packing virtual machines onto servers.

In the second paper, we investigate a formal model for packing virtual machines onto servers, along with how system administrators can adapt algorithms to this packing problem. We recognize that VM loads can be described probabilistically. One problem in extending the Bin Packing Problem to the Virtual Machine Assignment Problem is that items in bins have a deterministic weight, but VM loads are probabilistic. Estimations from the probability

distribution of VMs can be used to determine a deterministic number for item weights in the Bin Packing Problem. If loads can be represented with a probabilistic distribution, then system administrators can use probabilistic distributions to pack virtual machines efficiently. We show how system administrators can use these probabilistic distributions to carry out packings.

In the third paper, we investigate the effect of variance on these load distributions. We show that RGGA performs well for problems where virtual machines have low variance in their loads and outperforms other benchmarks. However, for problems where virtual machines have high variance in their loads, RGGA performs similarly with other algorithms.

Lastly, the research presented in this thesis focuses on predictively packing virtual machines onto servers, given distributions of loads on virtual machines. However, it is also possible, to some extent, to retrospectively pack virtual machines onto servers. Live migration allows a server administrator to move a running virtual machine or application between different physical machines without disconnecting the client or application. In the last part of this thesis, we wish to investigate the place live migration has with the problem of predictively packing virtual machines onto servers. We include a fourth chapter that investigates how live migration could affect the results presented in this thesis. We show that even though for some configurations, even if live migration is an option and swaps are cost-free, it still is important to do a good initial packing.

Chapter 2

Solving the Multi-Resource Bin Packing Problem with a Reordering Grouping Genetic Algorithm

To be submitted to GECCO-2011.

Data center energy costs, totalling $2.7 billion in the U.S. in 2005, can be reduced by utilizing virtual machines (VMs). Using virtualization, multiple operating system instances can be run on the same physical machine thus saving the power and associated cost of running separate server hardware for each service provided in a data center. To maximize the savings, administrators should pack as many VMs as possible onto a server while satisfying resource constraints. The packing of a set of VMs onto a set of available hardware can be seen as a type of bin packing problem.

We formally define multi-resource bin packing, a generalization of conventional bin packing, and develop an algorithm called Reordering Grouping Genetic Algorithm (RGGA) to assign VMs to servers. We first test RGGA on conventional bin packing problems and show that it yields excellent results but much more efficiently. We then generate a multi-constraint test set, not solvable by current methods, and demonstrate the effectiveness of RGGA in this context.

2.1 Introduction

In 2005, data centers in the United States accounted for 1.2% of all U.S. energy consumption, costing $2.7 billion [12]. Part of the problem is that most servers and desktops are in use only 5-15% of the time they are powered on, yet most x86 hardware consumes 60-90% of normal workload power even when idle [26, 4].

Data center costs can be reduced by utilizing virtual machines (VMs). Using virtualization, multiple operating system instances can be put on the same physical machine in order to more fully exploit hardware capabilities. System administrators can consolidate many VMs on the same physical machine, saving money. To maximize the savings, administrators should pack as many VMs as possible onto a server while satisfying certain performance criteria. We refer to this problem as *Virtual Machine Packing*.

VM packing can be seen as a type of bin packing problem. The bin packing problem is defined as follows: Given a set of items, each with its own weight, find the assignment of items to bins under which the number of bins used is minimized, without violating capacity constraints [3]. However, VM packing is not as simple as the conventional bin packing problem. Packing VMs onto servers is different in that each server has multiple types of constrained resources which the VMs consume. Each VM will add a set amount of load to different resources of the server, such as memory, disk space and CPU. Virtual machine packing is an example of one class of bin packing problems where each bin has multiple capacities and each item has multiple weights. The sum of the weights for any given resource must be less than or equal to the corresponding capacity. We call this the *Multi-Capacity Bin Packing Problem* (MCBPP) and define it formally in Section 2.2.1. We propose that the Multi-Capacity Bin Packing Problem can potentially be used to model the Virtual Machine Packing Problem.

Bin Packing is NP Hard [3]. Because of the difficulty of the problem, numerous approximation algorithms have been proposed including genetic algorithms (GAs).

We propose a new genetic algorithm to solve the MCBPP. Our new algorithm, the Reordering Grouping Genetic Algorithm (RGGA) is based on multiple representation of individuals in the GA. Multiple representations for each individual in the population are used in order to take advantage of an assortment of genetic operators. This modification allows RGGA to increase production of promising solutions and waste less time producing infeasible solutions.

Our paper is outlined as follows. Section 2.2 describes the bin packing problem and heuristics for solving it, including the first fit heuristic. Section 2.2.1 describes the MCBPP. Section 2.3 then describes how RGGA works with its new combination of genetic operators. In section 2.4, we will discuss our experimental setup. Finally, section 2.5 will discuss and analyze the results of our experiments.

2.2 Bin Packing

Definition 1. The *bin packing problem* is formulated as follows. Given a finite set of n items $I = \{1, 2, \ldots, n\}$ with corresponding weights $W = \{w_1, w_2, \ldots, w_n\}$ and a set of identical bins each with capacity $\mathcal{C}$, find the minimum number of bins into which the items can be placed without exceeding the bin capacity $\mathcal{C}$ of any bin. A solution to the bin packing problem is of the form $B = \{b_1, b_2, \ldots, b_m\}$, where each b_i is the set of items assigned to bin i, and is subject to the following constraints:

1. $\forall i \, \exists! \, j$ such that $i \in b_j$ (Every item has to belong to some unique bin.)
2. $\forall j \, \sum_{n \in b_j} w_n \leq \mathcal{C}$ (The sum of the weights of items inside any bin cannot be greater than the bin capacity.)

Definition 2. An *optimal packing* B to the bin packing problem is one where the particular assignments of items to bins minimizes the number of bins $|B|$.

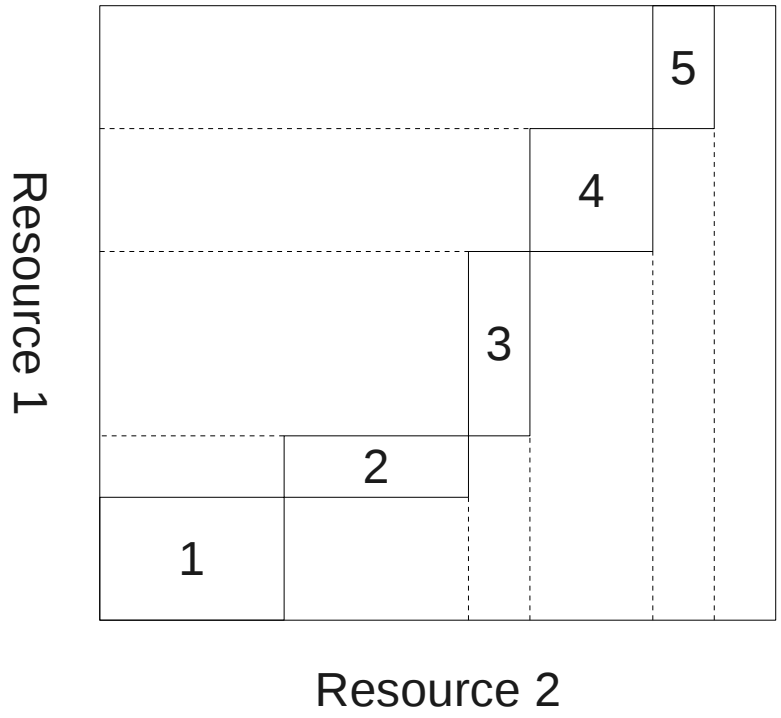

Figure 2.1: The two-capacity bin packing problem. The items in each bin are packed regardless of order. Resources used by each item are additive in each dimension.

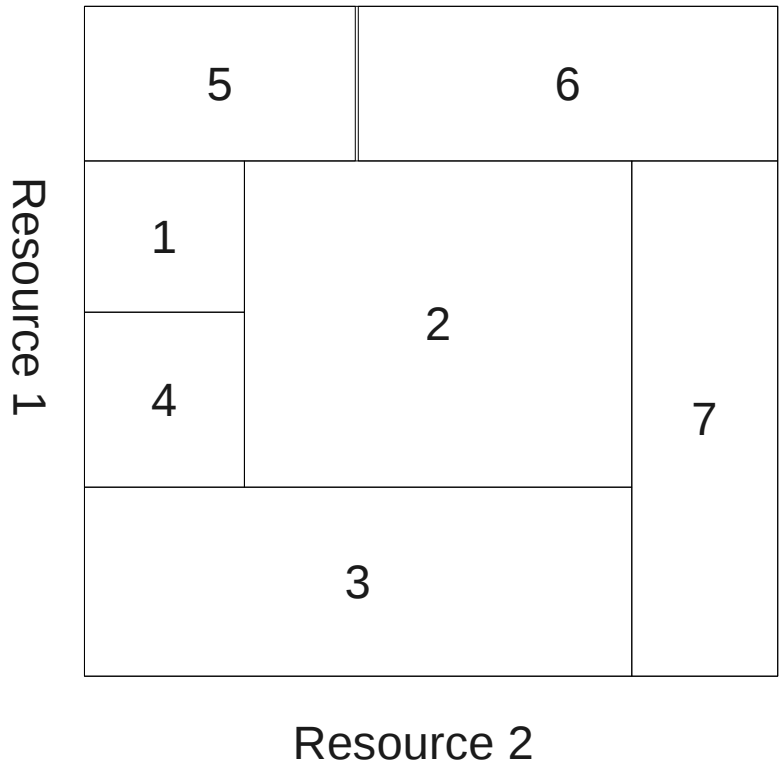

Figure 2.2: The conventional two-dimensional bin packing problem. The position and orientation of items in a bin matter.

2.2.1 Multi-Capacity Bin Packing

The Multi-Capacity Bin Packing Problem (MCBPP) is different from the conventional multi-dimensional bin packing problem, where the arrangement of items within a bin makes a difference in the end solution. In MCBPP, each dimension is associated with a specific resource; weights are summed independently in each dimension and cannot exceed any resource limit [24, 13]. An example of the two-capacity bin packing problem can be found in Figure 2.1 in contrast to the two-dimensional bin packing problem in Figure 2.2.

Definition 3. The *Multi-Capacity Bin Packing Problem* has the same definition as the conventional bin packing problem in Definition 2.2 with some variations. The capacity is a d-dimensional vector $\mathcal{C} = \langle C_1, C_2, \ldots, C_d \rangle$ where d is the number of resources. The weights are redefined so that the weight of item i is a d-dimensional vector $\boldsymbol{w}_i = \langle w_{i,1}, w_{i,2}, \ldots, w_{i,d} \rangle$.

1. $\forall i \, \exists! \, j$ such that $i \in b_j$ (Every item has to belong to some unique bin.)
2. $\forall j \, \forall k \, \sum_{n_k \in b_j} w_{n,k} \leq C_k$ (The sum of all the weights for any capacity for any bin must be less than the corresponding capacity for that bin.)

2.2.2 Solutions to the Bin Packing Problem

There have been numerous approximation algorithms to the conventional bin packing problem. In this section, we will cover the first fit heuristic, genetic algorithms, and other solutions for bin packing.

The first fit algorithm for bin packing is one of the most common heuristics for the problem. The algorithm processes items in arbitrary order. For each item, it attempts to place the item in the first bin that can accommodate the item. If no bin is found, it opens a new bin and puts the item within the new bin. One strength of the first fit heuristic is the running time. It is guaranteed to run in $O(n * log(n))$ where n is the number of items to be packed.

Permutation Pack (PP) attempts to find items in which the largest w components are exactly ordered with respect to the ordering of the corresponding smallest elements in the current bin [13]. For example, if $d = 2$ and $C_1 < C_2$, then we look for an item such that $w_1 < w_2$. If no item is found, the requirements are continually relaxed until one is found. One of the weaknesses of Permutation Pack is the running time. If all permutations are considered, it runs in $O(c!\, n^2)$ where c is the number of capacities and n is the number of items to be packed. We refer the reader to Leinberger et al. [13] for further description of the algorithm. In our experiments, we will use Permutation Pack for comparison.

Genetic Algorithms (GAs) offer another solution to the Bin Packing Problem. There is no rigorous definition for GAs [15]. GAs derive much of their inspiration from Darwinian biological processes. In GAs, individuals represent candidate solutions to the problem. These candidate solutions explore the solution space by undergoing processes similar to those of biological organisms. The simplest form of genetic algorithm involves three types of operators:

- **Selection**–Individuals in the population are selected for crossover with other individuals. Usually, selection is based on elitism, where the more fit individuals are selected more often than less fit individuals.

- **Crossover**–Two individuals in the population exchange subsections of their candidate solution with each other to create two offspring.

- **Mutation**–After crossover, each individual has a probability of having their candidate solution modified slightly (mutated).

These three operators act on the candidate solutions in the population in order to improve the population [15].

Radcliffe et al. [17] proposed six design principles of good representations for individuals in a genetic population. One of the design principles discussed was avoiding redundancy.

Avoiding redundancy in the representation allows the GA to search a wider space of possible candidate solutions with fewer computations. Falkenauer [7] discussed that for some types of problems, grouping genetic algorithms (GGAs) avoid redundancy well. In GGAs, the chromosome represents groups. GGAs have been used to solve many types of grouping problems, such as bin packing and graph coloring. One example of a GGA to solve the bin packing problem is Rohlfshagen et al's [18] exon shuffling genetic algorithm.

Other approaches to the bin packing problem include BISON [19], MBS′, and VNS. The first algorithm, BISON, applies a first fit decreasing strategy to the bins followed by a branch and bound procedure [19]. MBS′ attempts to find a set of items (packing) that fits the bin capacity as much as possible [8]. VNS explores increasingly distant neighborhoods of the current solution and jumps from there to a new one if and only if an improvement has been made [8].

Alvim et al. [3] developed a highly successful heuristic to the bin packing problem which combines multiple algorithms. Their algorithm, HI_BP, uses reduction techniques to eliminate some items and to fix the items in some bins. Afterward, lower bounds and upper bounds are computed. The algorithm halts at this stage if the lower and the upper bounds are equal. Next, a greedy algorithm is applied to build a solution using exactly the number of bins returned by the lower bound (this solution many times will be infeasible). If the solution that was constructed is not feasible, load redistribution strategies are employed. Next, if the solution still is not feasible, a tabu search heuristic is used to attempt to knock down capacity violations. If at this point, the solution is feasible, then the algorithm stops. If the solution is still not feasible, the algorithm adds 1 to the lower bound, and goes back to the reconstruction phase.

As noted by Rohlfshagen et al. [18], Alvim et al.'s approach "relies on several preprocessing steps and undergoes multiple phases exploiting several mathematical properties of the bin packing problem." This approach would not directly extend well to the MCBPP.

2.3 Reordering Grouping Genetic Algorithm

To solve MCBPP, we propose the Reordering Grouping Genetic Algorithm (RGGA). Because RGGA relies heavily upon the behavior of first fit packing, we prove here that there exists a first fit ordering which will produce an optimal packing for every instance of the bin packing problem. Note that the first fit algorithm as defined works for both the conventional bin packing problem and MCBPP.

Definition 4. A *packing* B is any feasible solution to a bin packing problem.

Definition 5. Given a packing B a *packing sequence* $\langle i_{1,1}, i_{1,2}, \ldots, i_{1,n_1}, i_{2,1}, i_{2,2}, \ldots, i_{2,n_2}, \ldots, i_{k,1}, i_{k,2}, \ldots, i_{k,n_k} \rangle$ is a sequence of items formed by a nested iteration over:

- all bins of B in arbitrary order
- and all items within each bin in arbitrary order.

This sequence is not unique—there may be many packing sequences for any given packing.

Definition 6. A *first fit packing* is obtained by executing the first fit algorithm on a packing sequence.

Theorem 2.3.1. *If B is a packing with $|B|$ bins, and $\mathcal{S}$ is any packing sequence of B with corresponding first fit packing $\mathcal{B}$, then $|\mathcal{B}| \leq |B|$, i.e. $\mathcal{B}$ has no more bins than B.*

Proof. Let $\mathcal{S}_k = \langle i_{1,1}, i_{1,2}, \ldots, i_{1,n_1}, i_{2,1}, i_{2,2}, \ldots, i_{2,n_2}, \ldots, i_{k,1}, i_{k,2}, i_{k,n_k} \rangle$ be the subsequence of $\mathcal{S}$ corresponding to the first k bins. Let $\mathcal{B}_k = \{\beta_1, \beta_2, \ldots \beta_k\}$ be the first fit packing of $\mathcal{S}_k$.

The first fit algorithm will assign $\mathcal{S}_1$ in order into bins in $\mathcal{B}$. Items $i_{1,1}, i_{1,2}, \ldots, i_{1,n}$ will fit into bin β_1 because these items were in a single bin in the packing B. Therefore, $|\mathcal{B}_1| \leq 1$.

By way of induction, suppose that the first fit packing of $\mathcal{S}_k$ uses at most k bins i.e. $|\mathcal{B}_k| \leq k$. We will show that the first fit packing of $\mathcal{S}_{k+1}$ uses at most $k+1$ bins i.e.

$|\mathcal{B}_{k+1}| \leq k+1$. The first fit algorithm applied on $\mathcal{S}_{k+1}$ will start by assigning the items in order from $\mathcal{S}_k$ into the first $|B_k|$ bins of $\mathcal{C}$. The remaining items $i_{k+1,1}, i_{k+1,2}, \ldots, i_{k+1,n_{k+1}}$ may also fit into extra space in the first $|B_k|$ bins. In any case, they will require at most one additional bin because they all fit in a single bin, specifically β_{k+1} in the packing B. Therefore, $|\mathcal{B}_{k+1}| \leq |\mathcal{B}_k| + 1 \leq k+1$.

By induction, $|\mathcal{B}| \leq |B|$. □

Corollary 2.3.2. *For any solvable bin packing problem instance, there exists a packing sequence whose first fit packing is optimal.*

Proof. Let B^* be an optimal packing for the packing sequence, let $\mathcal{S}^*$ be any packing sequence of B^*, and let $\mathcal{B}^*$ be the first fit packing of $\mathcal{S}^*$. By Theorem 2.3.1 $|\mathcal{B}^*| \leq |B^*|$. Because B^* is an optimal packing, $|\mathcal{B}^*| = |B^*|$ and $\mathcal{B}^*$ is optimal. □

Because there exists some first fit packing for every optimal packing, we can reduce MCBPP to a search for a packing sequence with an optimal first fit packing.

2.3.1 Algorithm Overview

Even though Radcliffe et al. [17] proposed that representations of individuals should avoid redundancy, and Falkenauer showed that GGAs represent the problem well, strictly following a grouping genetic algorithm approach is not flexible enough for MCBPP. For example, as will be seen in Section 2.5, the conventional grouping mutation operators do not perform well for MCBPPs. Rohlfshagen et al. [18] proposed a grouping mutation operator that, with equal probability, swaps two items in different bins or moves an item in one bin to another. When this operator is applied to MCBPP with just two capacities, the mutation operator described by Rohlfshagen et al. produced infeasible candidate solutions in over 95% of cases tested. This high percentage of infeasible candidate solutions wastes time and resources when solving the problem. Even though Falkenauer proposed a different, possibly more successful

mutation operator, this mutation failed to produce good results when the problem size was large. None of the grouping approaches to mutation seem to work well for MCBPP.

RGGA represents individuals in the population not only as an assignment of items to bins, but also as a packing sequence. In order to represent a candidate solution this way, there must be a function to transform a packing sequence to a packing and vice-versa. Using these transformations, we represent a candidate solution with multiple representations and are able to create operators that take advantage of multiple representations. This gives RGGA the power to use special genetic operators. Not only can RGGA use both ordering and grouping genetic operators, but it can also use operators that have both ordering and grouping components.

When an ordering operator is performed, an assignment of items to bins must be generated. Therefore, the items are first fitted into the solution in order to generate the assignment. When a grouping operator is performed, a packing sequence must be developed. We iterate over each bin and over each item inside of each bin, adding each item encountered to the packing sequence in order.

Because RGGA uses a first fit ordering, it naturally avoids generating infeasible solutions. Other bin packing algorithms need to conduct various searches in order to eliminate infeasible solutions [3]. Because RGGA's solutions are defined as the first fit packing of the list of items, and the first packing will always yield a feasible solution, every solution returned by RGGA will be feasible. This eliminates the time that would be spent cutting down infeasibilities.

2.3.2 Fitness Function

Every GA needs a fitness function to evaluate the fitness of an individual in the population. For the conventional bin packing problem, Falkenauer [7] proposed the following cost function of a packing B:

$$f_{BPP}(B) = \frac{1}{|B|} \sum_{j=1}^{|B|} F(b_j) \tag{2.1}$$

$$F(b_j) = \left(\frac{1}{C} \sum_{i \in b_j} w_i \right)^{\kappa} \tag{2.2}$$

where w_i is the weight of item i, C the bin capacity, and κ a constant, $1 < \kappa \leq 2$. We use Falkenauer's suggestion of setting $\kappa = 2$.

We adapt this fitness function to MCBPP by incorporating multiple capacities in the fitness function. We first define $F_k(b_j)$ to be the normalized sum of weights if items in bin j and resource k:

$$F_k(b_j) = \left(\frac{1}{C_k} \sum_{i \in b_j} w_{i,k} \right)^{\kappa} \tag{2.3}$$

where $w_{i,k}$ is the weight of item i with respect to resource k, C_k the bin capacity for resource k, and κ a constant, $1 < \kappa \leq 2$.

We next define $f_{MBPP,j}(B)$ by using equation 2.1 for one single resource:

$$f_{MBPP,k}(B) = \frac{1}{|B|} \sum_{j=1}^{|B|} F_k(b_j) \tag{2.4}$$

We define $f_{MBPP}(B)$ by summing $f_{MBPP,k}(B)$ for all resources:

$$f_{MBPP}(B) = \sum_{k=1}^{d} f_{MBPP,k}(B) \tag{2.5}$$

This function, $f_{MBPP}(B)$, is the fitness function of a single individual B in the genetic population. We must assess whether this metric will always be minimized together with the optimal packing.

Theorem 2.3.3. *Given a multi-capacity bin packing problem, if B^* is an optimal packing and B' is a packing the same items as B^* such that $|B^*| < |B'|$, then $f_{MBPP}(B^*) < f_{MBPP}(B')$.*

Proof. The conventional bin packing problem is a special case of the multi-capacity bin packing problem in that for any fixed k, $f_{MBPP,k}$ is equivalent to f_{BPP}. This, along with a

theorem in Falkenauer [7], implies that for any k, $f_{MBPP,k}(B^*) < f_{MBPP,k}(B')$. Therefore,

$$\begin{aligned} f_{MBPP}(B^*) &= \sum_{k=1}^{d} f_{MBPP,k}(B^*) \\ &< \sum_{k=1}^{d} f_{MBPP,k}(B') \\ &= f_{MBPP}(B') \end{aligned}$$

□

2.3.3 Crossover Operator

Crossover in genetic algorithms is where individuals in the population combine traits in order to generate a new generation. The child theoretically should have good traits from both parents and hopefully has better fitness.

We choose two parents with probability proportional to the fitness of the individual. In other words, more fit individuals will reproduce with higher probability than less fit individuals.

The crossover operator is implemented using an exon shuffling approach similar to the one described by Rohlfshagen et al. [18]. This approach combines all of the bins from both parents and sorts the bins by fitness. The more full bins are at the front of the list, while the less full bins are at the end. The algorithm systematically picks the more full bins and keeps them intact. If any bin picked contains any item that belongs to a bin that has already been picked, that bin is discarded. This process will produce a list of bins that may not include all items. These remaining items that have not been included in any bin are then sorted in decreasing order and the first fit decreasing heuristic is applied to these items.

Because RGGA uses multiple representations, the resulting assignment of items to bins must be transformed into a packing sequence for other operators. This packing sequence, when first fit packed, may generate a different set of bins. By theorem 2.3.1, we know that

the resulting assignment of items to bins after applying the first fit operation will require no more bins than was required before. The only case where the structure of the items is not preserved is in the case where an item from a later bin is packed into an earlier bin. We consider this a benefit. Because the crossover operator sorts the bins in order of fitness, the earlier bins are generally more tightly packed than the later bins. If the algorithm is able to move any items from later, less tightly packed bins to earlier, tighter bins, the change would be welcome as the later, looser bins now have more space to rearrange.

The operator described here is inherently greedy in that it considers the most tightly packed bin first. Therefore, in order to escape from local optima, we mimic Rohlfshagen et al. [18] by adding noise to the fitness of each bin. The noise of each bin is distributed as a Gaussian with mean $\mu = 0$ and standard deviation $\sigma = \frac{\mathcal{C}}{10}$ where $\mathcal{C}$ is the bin capacity. When bins have multiple capacities, $\sigma = \sum_{i=1}^{|\mathcal{C}|} C_i/10$ where $\mathcal{C}$ is the set of all bin capacities for that bin, and C_i is member i of that set.

2.3.4 Mutation Operator

RGGA's mutation operator includes three options. First, Falkenauer proposed a mutation operator where a number of bins are removed from the solution and the items are subsequently reinserted into the candidate solution. Second, two items in the packing sequence of items are swapped. Third, one item is moved to another spot in the packing sequence.

For the second and third genetic operators that work on the packing sequence list, we use information obtained from the groups to improve the performance of the ordering genetic operators. For example, we assure that we never swap two items that came from the same bin. Also, we do not move an item in one bin to the spot of another item from the same bin. Lastly, for all three genetic operators, we mutate items with probability inversely proportional to the fitness of the bin that the item comes from. Items from worse bins are mutated more often than items from better bins. This helps to assure that the structure of better bins is maintained.

We will discuss differences in performance between the three mutation operators in section 2.5. However, for testing RGGA, we settled on applying Falkenauer's remove mutation operator with probability $\frac{1}{3}$, swapping two items with probability $\frac{1}{3}$ and moving an item with probability $\frac{1}{3}$.

2.4 Experimental Setup

The GA used for all the experiments was a standard steady-state GA with a population size of 75. The previously discussed crossover operator was applied with probability 0.8. In the event that a crossover is not applied, both parents have a uniform probability of being reproduced asexually. The previously discussed mutation operator was applied with probability 0.1 to each individual after crossover.

Naturally, any new algorithm needs to be compared against existing algorithms. However, because MCBPP described in this paper is new, there are no standard benchmark problems to use for comparison. Therefore, we will test the performance of RGGA in two parts. First, we specify the number of capacities on MCBPP to be one thus reducing it to a conventional bin packing problem. This will allow us to test against conventional evolutionary approaches. Second, we will evaluate the performance of RGGA on a set of MCBPPs generated by us.

2.4.1 Conventional Bin Packing Problem

When comparing RGGA to other conventional bin packing algorithms, we will use a set of benchmark instances produced by A. Scholl and R. Klein. The third set, labeled HARD0–HARD9, is a set of triplets. The data set contains 10 problems, each consisting of 200 items with weights between 20,000 and 35,000 and bin capacities of 100,000. We will focus our comparison against other bin packing algorithms using this data set as it is deemed the most difficult. As we will compare RGGA against Exon Shuffling Genetic Algorithm [18], we

follow Rohlfshagen et al.'s precedent and execute our algorithm 50 times on each instance, with a maximum limit of 50,000 function evaluations.

2.4.2 Multi-Capacity Bin Packing Problem

The goal of this paper is to develop a solution to MCBPP that can be applied to pack VMs onto servers. We developed a set of MCBPPs that model the number of VMs normally found on servers. Based on studies by VMware [27], we generated a test set of MCBPPs that have close to 9-13 items per bin[1]. This test set of multi-capacity bin packing instances is intended to simulate packing VMs onto a cluster of servers. For the purposes of this experiment, we packed each process according to two different resources: processor time and RAM used. We assigned loads to VMs such that nine to thirteen of the VMs can fit on a physical server. The data sets that we used were created such that every item fits perfectly into the set of bins (there is no free space in any bin). We created 10 different two-capacity problem instances of differing size. The smallest problem has approximately 500 VMs while the largest problem has approximately 4500 VMs. An optimal bin packing is extraordinarily difficult to find because each problem was built to have no free space in any of its bins.

Our algorithm was executed 50 times on each problem instance. We initially used a maximum limit of 50,000 function evaluations. However, after analyzing the convergence of the algorithm, we test our algorithm using 7500 function evaluations instead. We report the average number of bins found in the best solution along with the average, minimum and maximum number of function evaluations.

2.4.3 RGGA For Virtual Machine Deployments

In order to give validity to this work, we used RGGA to pack real VMs onto servers. We benchmarked RGGA against the algorithms First Fit and Worst Fit. First Fit has been described in Section 2.2.2. The Worst Fit algorithm for bin packing considers each item in

[1] `http://aml.cs.byu.edu/~davidw/rgga/datasets`

Approach	Solved
Relaxed MBS' [8]	2
VNS [8]	2
Perturbation MBS' & VNS [8]	2
GA [14]	3
BISON [19, 14]	3
Dual Tabu [19, 8]	3
Exon Shuffling GA [18]	8
RGGA	**9**

Table 2.1: Our algorithm compared with other algorithms on the same 10 hard conventional bin packing problem instances. These instances are not representative of MCBPP.

order. For each item, it considers only bins that already have at least one item placed in them. It places the item in the bin into which it fits which has the most amount of free space. If the item does not fit in any of the bins considered, it is placed into a new bin.

In our experiments, we created VMs which would have known and deterministic load. We realize that this case does not happen in real deployments, but defer this issue for future research. We created 50 VMs which exhibited a deterministic load on their respective servers. We allowed the packing algorithms to see the loads that these machines exhibit. Then, we used the kvm hypervisor and the qemu monitor to create real VMs on our respective servers. After the VMs had booted, we assigned loads to these VMs and observed resource utilization.

The hardware that we used for benchmarking contain Intel quad core 2.66 gigahertz cpus with cache size of three MB and a RAM size of 8 GB.

2.5 Results

2.5.1 Conventional Bin Packing Problem

We intend to show that RGGA performs competitively with the best genetic algorithms on single capacity bin packing problems and that it performs well on MCBPPs. Table 2.1 shows a comparison of several approaches in recent literature on the same set of 10 benchmark instances of one-dimensional bin packing.

instance	Opt	RGGA Avg.	Exon Shuffling Avg
HARD3	55	55.96	56
HARD4	57	57	57.02
HARD5	56	56	56.06
HARD9	56	56	56.02

Table 2.2: Summary of RGGA compared with Rolfshagen et al.'s Exon Shuffling approach.[3]

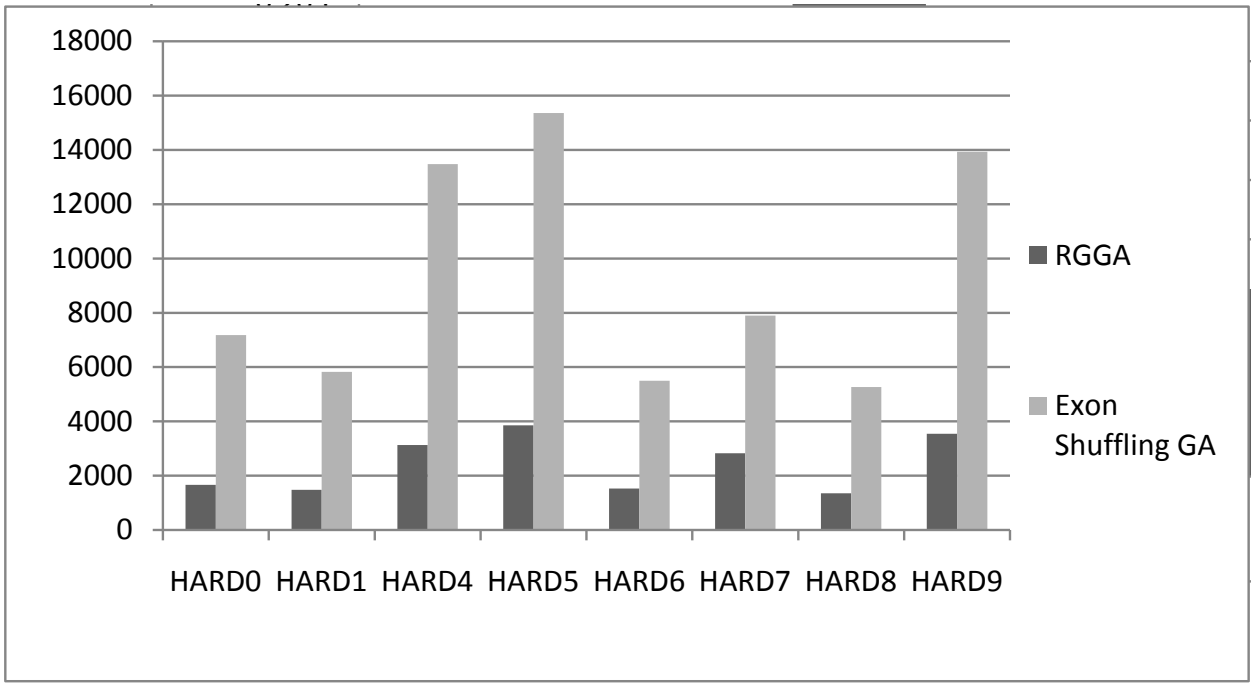

Figure 2.3: The average number of function evaluations for the exon shuffling GA are compared with the function evaluations for RGGA.[5]

In most respects, RGGA seems superior. RGGA solves HARD3 to optimality 4% of the time, while exon shuffling genetic algorithm never solves it. For some of the problems that exon shuffling genetic algorithm did not solve completely all the time, such as HARD5, RGGA solves to optimality every time.

Even though RGGA performed slightly better in terms of percent of problems solved, it performed much better in terms of function evaluations. Figure 2.3 shows the average number of function evaluations for RGGA and exon shuffling genetic algorithm for different

[3]HARD0, HARD1, HARD2, HARD6, HARD7 and HARD8 are omitted because both algorithms performed equally well on those problems

instance	Avg FE	Min FE	Max FE
HARD0	2462.9	2488	3394
HARD1	1877.8	2384	2932
HARD2	50000	50000	50000
HARD3	47955.84	8453.0	50000
HARD4	4094.6	6362	6587
HARD5	5337.0	6316	9295
HARD6	2058.6	1924	3502
HARD7	3760.9	2275	5246
HARD8	1934.5	1830	3073
HARD9	4637.8	5070	7977

Table 2.3: The average, minimum and maximum number of function evaluations for RGGA on the single-capacity data set used.

instance	Avg FE	Min FE	Max FE
HARD0	7177.38	2488	17186
HARD1	5826.58	2384	15996
HARD2	50000	50000	50000
HARD3	50000	50000	50000
HARD4	13474.10	6362	35809
HARD5	15361.15	6316	44610
HARD6	5501.98	1924	10143
HARD7	7899.92	2275	22461
HARD8	5265.58	1830	21475
HARD9	13931.55	5070	30045

Table 2.4: The average, minimum and maximum function evaluations for exon shuffling genetic algorithm on the single-capacity data set used.

problems. RGGA performs significantly better in this regard. See also Tables 2.3 and 2.4 for more details.

2.5.2 Multi-Capacity Bin Packing Problem

For the multi-capacity problem, we tested RGGA using the multi-capacity data set described in Section 2.4. Note that no other algorithms discussed in this paper except for the baseline solve this problem, and therefore cannot be used for comparison. RGGA is meant to solve MCBPPs like this data set rather than instances of the conventional single-capacity bin

[5]HARD2 and HARD3 are omitted because of graph scaling. Both algorithms run approximately to the 50000 function evaluation limit. See Tables 2.3 and 2.4

Opt	RGGA Found	FFD Found	PP Found
59	60.00	61	61.4
112	113.00	121	117.8
191	192.00	207	196.5
216	217.00	234	223.9
241	242.00	262	250.3
267	268.00	296	277.7
320	321.00	353	331.8
371	372.00	412	384.7
425	426.02	465	439.8
481	482.00	541	499.5

Table 2.5: Summary of our experimental results of our algorithm on the multi-capacity problems.

packing problem. The items in this data set represent VMs and the bins in this data set represent servers. These problems were meant to be very hard as the optimal packing will have no free space in any bin. Figure 2.5 shows that RGGA was always one bin away from the optimal packing in every run, except for one run when the optimal number of bins was 425.

Also, Figure 2.5 shows a comparison of RGGA, Permutation Pack (PP) and the first fit decreasing (FFD) heuristic for each problem instance. Because MCBPP has multiple weights for each item, in order to compare two different items in MCBPP, we use the following scoring, $S(i) = \sum_{j=1}^{k} w_{i,j}^2$ where i is a particular item, $S(i)$ is the score of the item, k is the number of weights belonging to item i, and $w_{i,j}$ is weight j of item i. RGGA significantly outperforms the first fit decreasing heuristic.

In order to show that a grouping representation is not desired all of the time for all genetic operators, we show that the ordering mutations outperform the grouping mutations for large MCBPPs (the type of problems that this algorithm would encounter when packing VMs onto servers). In order to show that RGGA's ordering approach to the mutation operator is an improvement upon Falkenauer's grouping approach to mutation for large multi-capacity problems, we compared the percentage of mutations that led to improved candidate solutions for the three different types of mutations that RGGA used. Recall that

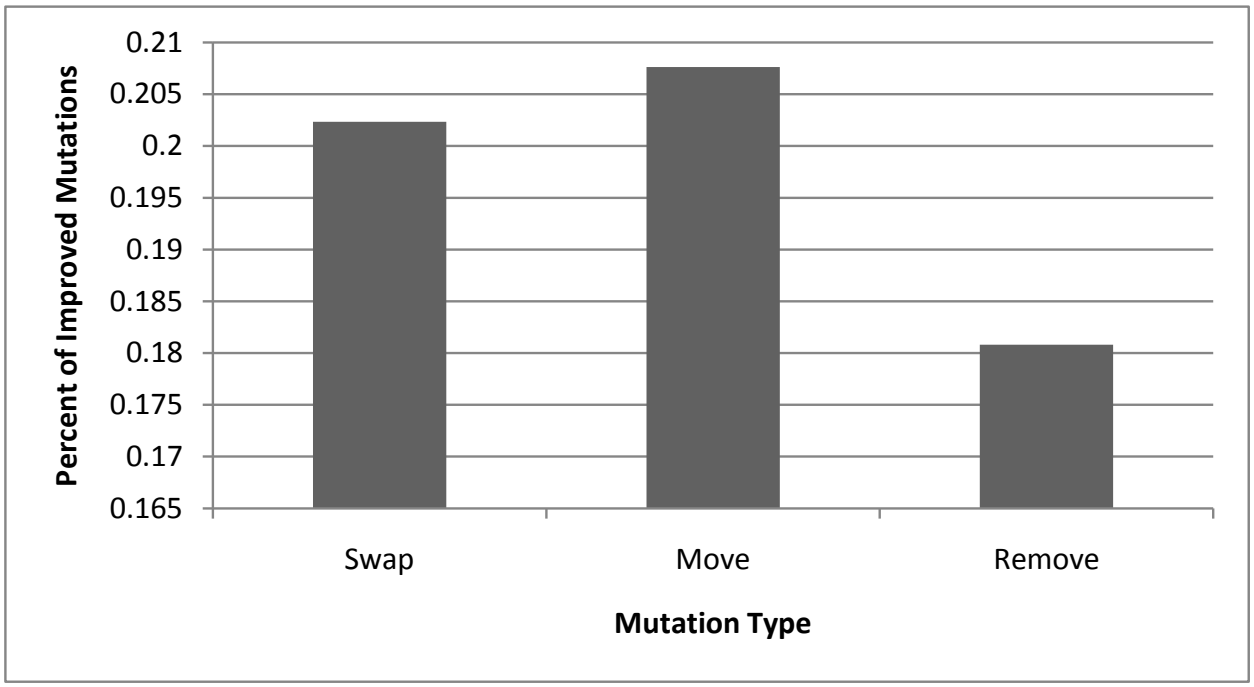

Figure 2.4: The average improved solutions for each mutation type for the multi-capacity problems.

RGGA used three different mutation operators: remove a few bins and reinsert their contents (remove), move an item in the ordered item list (move), and swap two items in the item list (swap).

We generated a new test set of multi-capacity bin packing instances of corresponding to cases with the number of items between 500 and 4500. We performed this test for problems of varying numbers of items. We show a summary of our findings in Figure 2.4 for MCBPPs. The graph shows that the move mutation outperformed Falkenauer's remove mutation. Move and swap mutations improved their solutions in about 20% of cases. Falkenauer's remove mutation improved its solutions in about 16% of cases. A more detailed graph of how each operator did on each specific problem size can be found in Figure 2.5. As can be seen in Figure 2.5, all of the mutation operators performed nearly equally well for small data sets. However, when the size of the data set increases, the performance of Falkenauer's remove mutation operator diminishes while the move and swap mutation operators still perform fairly well.

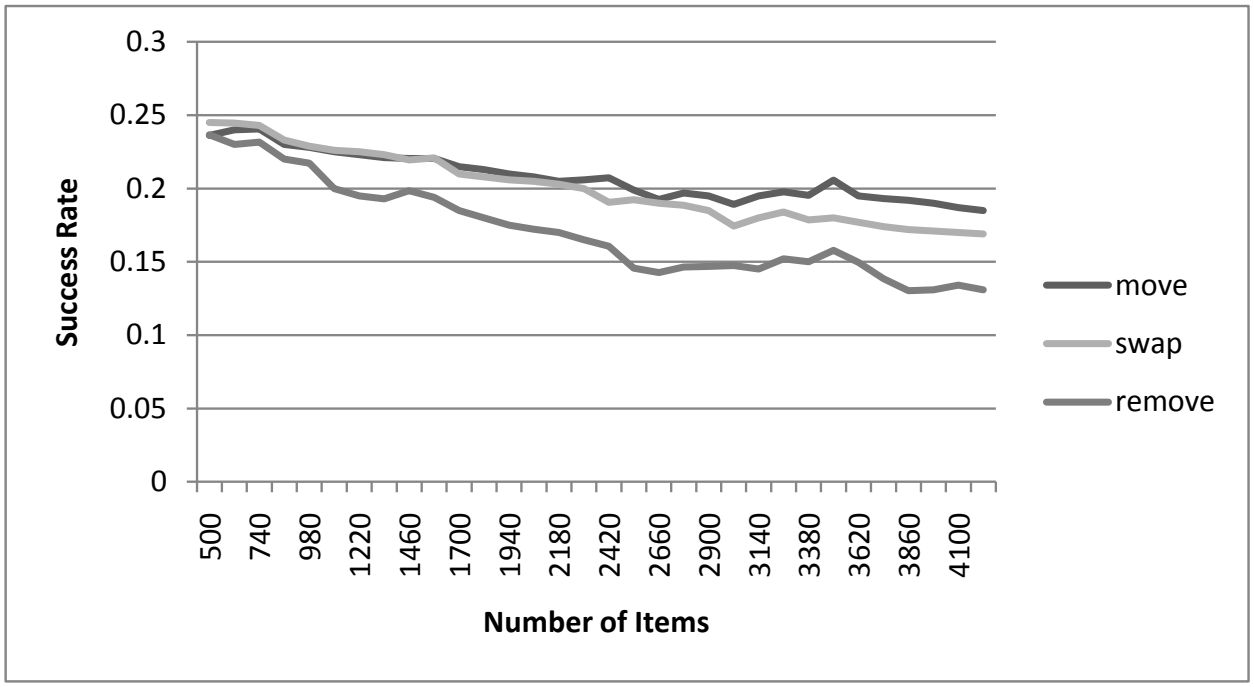

Figure 2.5: The average percent of mutations that improved individuals plotted against the number of items in each of MCBPPs.

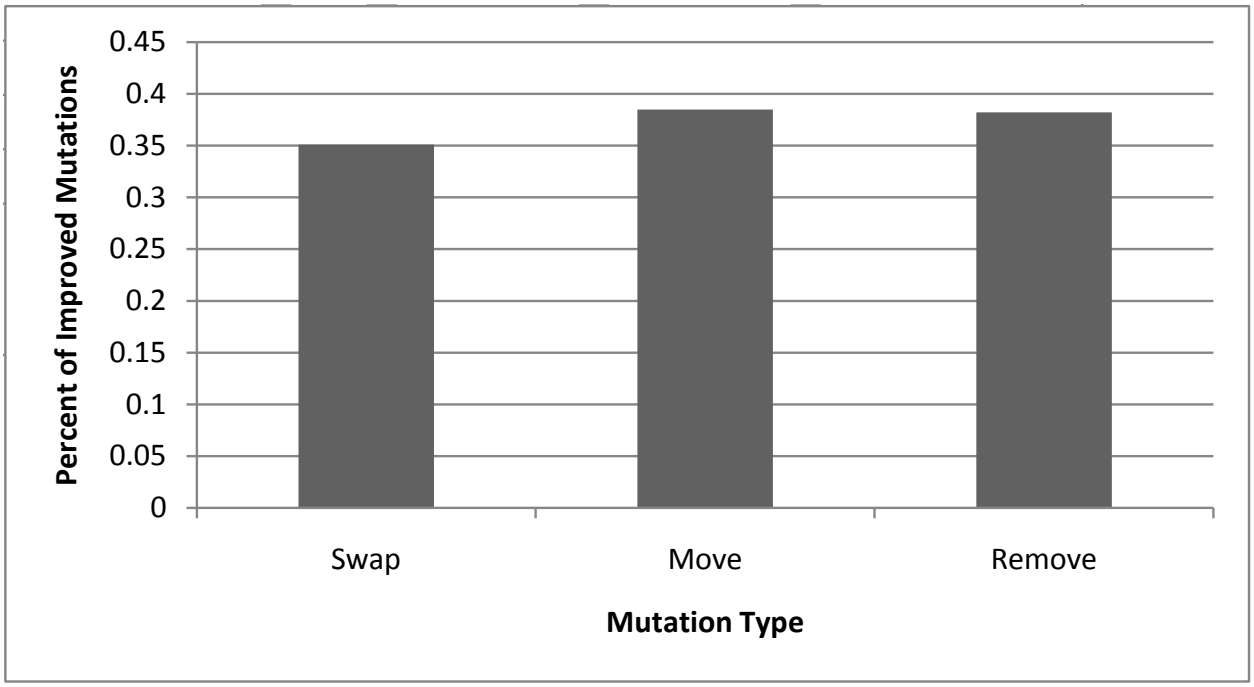

Figure 2.6: The average percent of mutations that improved individuals for single capacity problems.

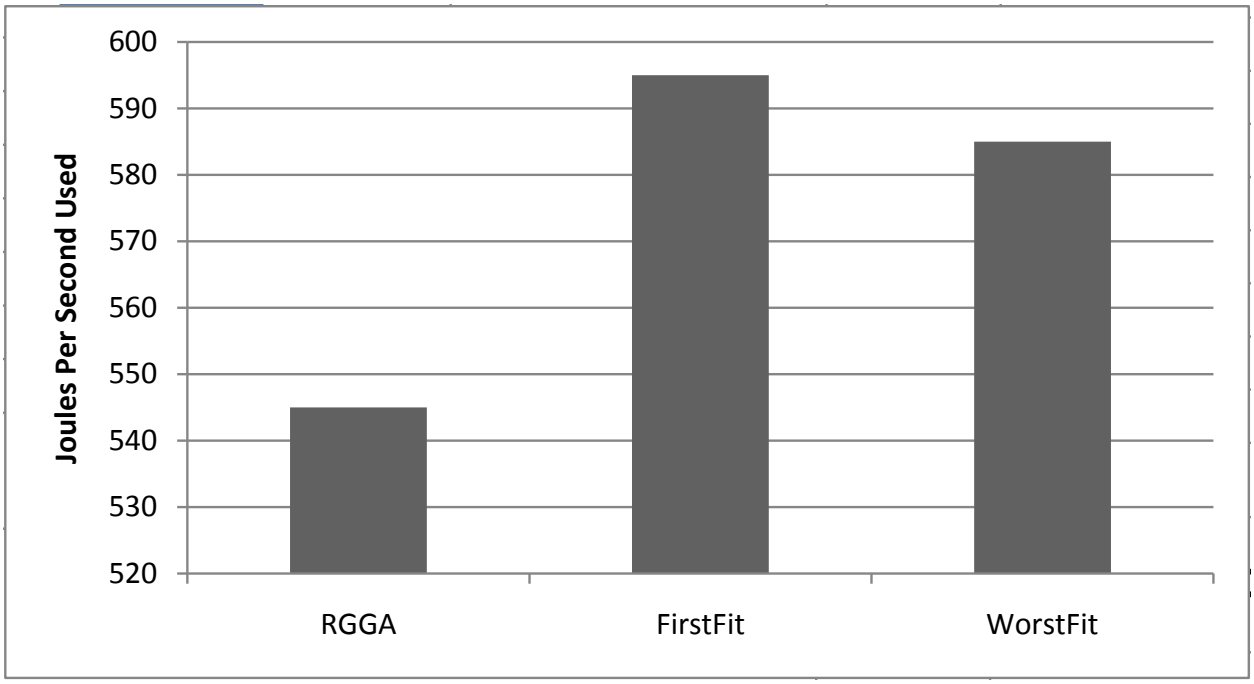

Figure 2.7: The amount of energy used in solutions returned by the three different algorithms.

A comparison of each mutation's performance for single-capacity problems can be found in Figure 2.6. For single-capacity problems (HARD0–HARD9), the move and swap mutations did not perform as well in comparison with the multi-capacity problem. The swap mutation performed worse, but did not perform substantially worse than the move mutation and Falkenauer's mutation. It may be that these test problems were small enough that Falkenauer's mutation still performed reasonably well.

2.5.3 RGGA For Virtual Machine Deployments

After creating artificial loads on VMs, we observed the total amount of energy used by all servers used. We show in Figure 2.7 the energy used by the deployments created by the three different algorithms. The deployment created by RGGA uses much less energy than the other benchmarks we used. If loads on virtual machines are deterministic and known, RGGA can take advantage of this, and pack VMs tightly onto servers.

In the Sections 2.5.1 and 2.5.2 we showed that RGGA is very good at finding good answers. We show here that RGGA also converges very quickly, thus limiting the running

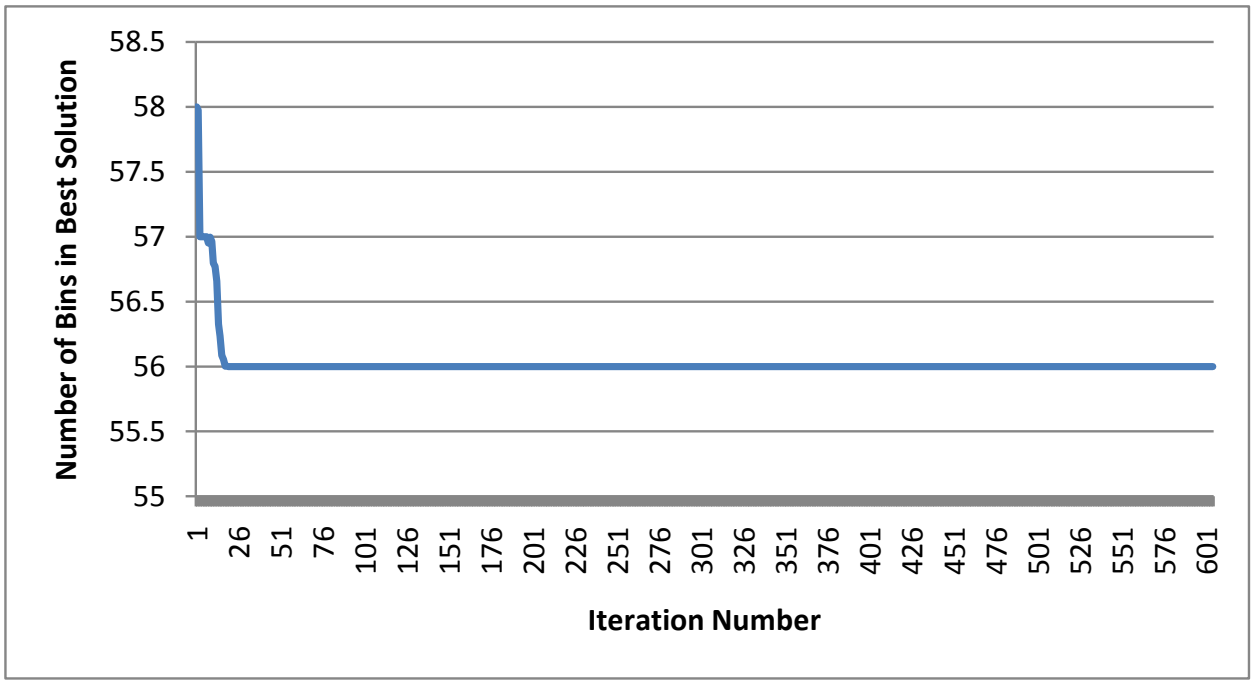

Figure 2.8: The average number of bins in the best solution for different iteration numbers for HARD0.

time needed when it is used in data centers. We show in Figure 2.8 the average number of bins in the best solution in the population at varying iteration numbers when running RGGA on HARD0. As can be seen, RGGA finds an optimal solution in nearly every case in fewer than thirty iterations.

We wished to find out how much real time RGGA would take to run in a real-life situation. We imagine that there may be problem sets which may be harder to optimize than these problems here, and thus give RGGA a maximum of 100 iterations or 7500 function evaluations instead of thirty iterations or 2250 iterations. We benchmarked the time RGGA needed to find come to a solution after 100 iterations for varying sizes of Multi-Capacity Bin Packing Problems.

We compared the number of servers found for all ten multi-capacity problems used for 50,000 function evaluations and for 7500 function evaluations. In all ten multi-capacity problems, there was no difference in the number of servers found in the solution between

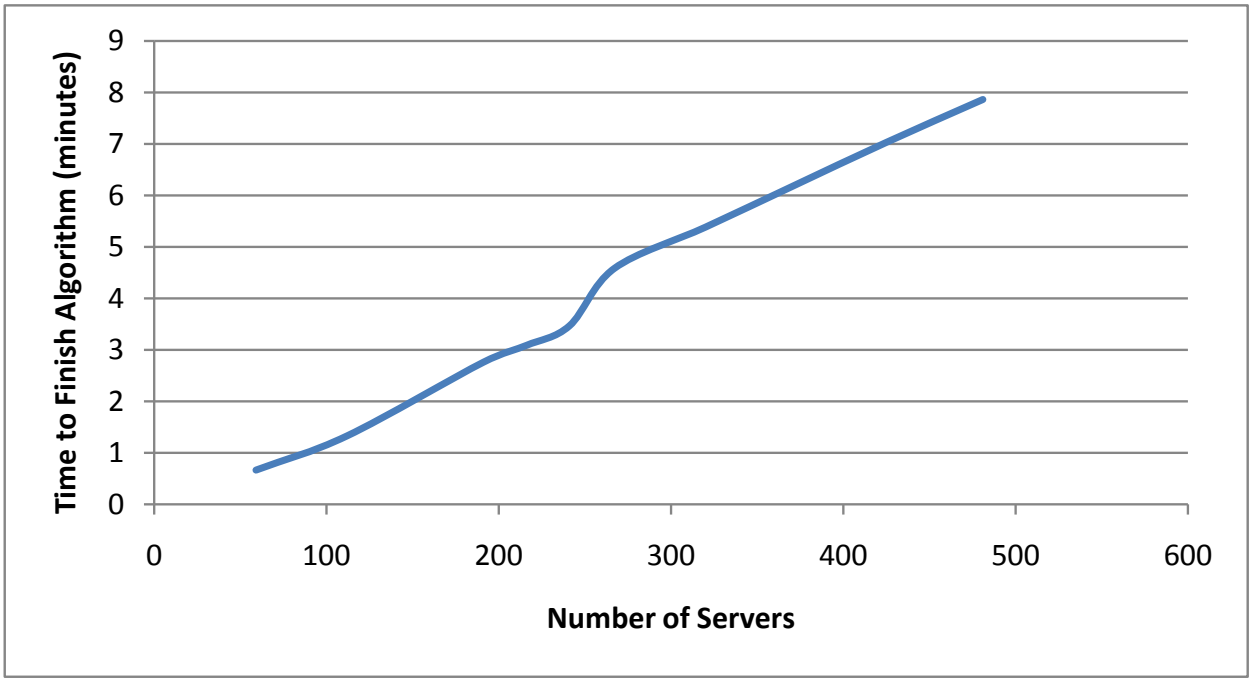

Figure 2.9: The amount of time taken to find a solution for varying sizes of Multi-Capacity Bin Packing Problems.

50,000 function evaluations and 7500 function evaluations. This gives validity to the data we show about the practical amount of time taken to solve RGGA.

We show in Figure 2.9 the amount of time needed to solve RGGA on our hardware. The figure shows RGGA scales linearly as the problem size increases. Moreover, data centers of even 500 servers can be solved in fewer than 10 minutes on a single core. We imagine that most data centers with 500 servers could spare a single core for ten minutes without much trouble in order to optimize the virtual machines more efficiently. This shows that the running time of RGGA is generally feasible to run in data centers.

Finally, we compared the number of servers found for the different algorithms to see which algorithms, in general, found solutions with fewer servers. Figure 2.10 shows the number of servers found for the three different algorithms being compared. Although RGGA performs better than the other two algorithms, PP also performs well.

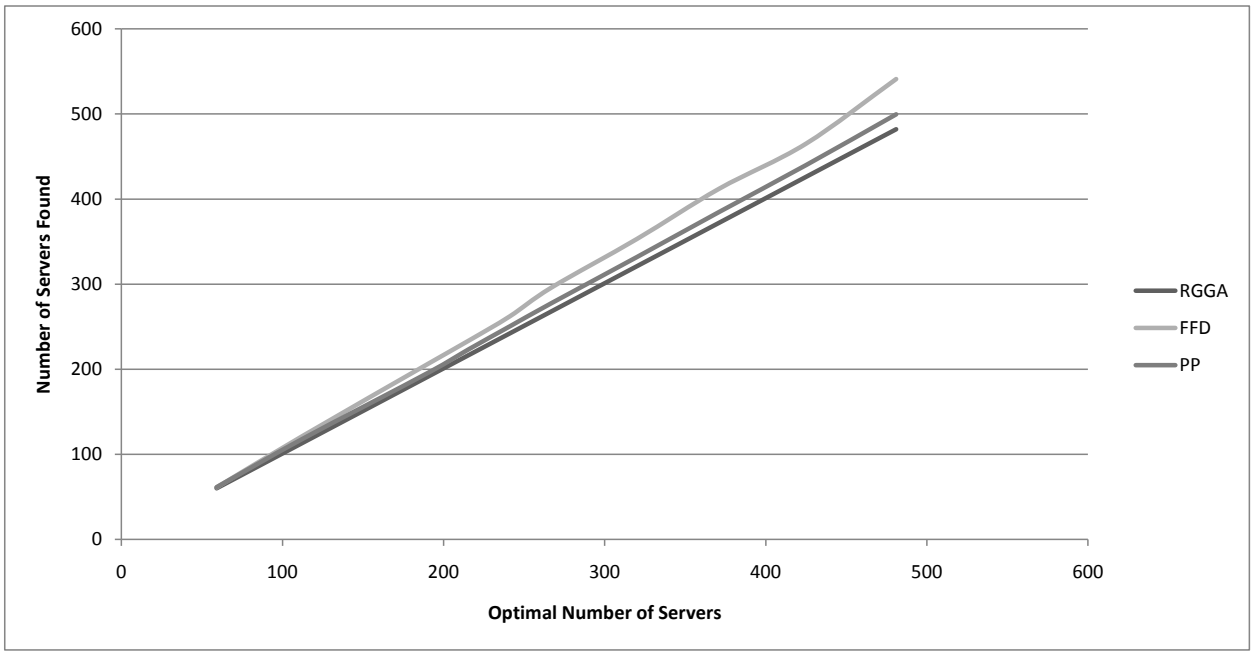

Figure 2.10: The number of servers found by solutions of RGGA, First Fit Decreasing (FFD) and Permutation Pack (PP).

2.6 Conclusions

A novel genetic algorithm is proposed to solve the bin packing problem. A way to generalize the genetic algorithm to the bin packing problem is shown. the genetic algorithm is shown to outperform existing evolutionary approaches in the literature, as well as to perform well on the new generalization of bin packing. This new Genetic Algorithm has been shown to perform well on the Virtual Machine Assignment Problem.

Chapter 3

Probabilistic Virtual Machine Assignment

Accepted to Cluster Computing 2010.

We cast the assignment of virtual machines (VMs) to physical servers as a variant of the classic bin packing problem. We then develop a probabilistic model of VM load that can be used to produce assignments of VMs to servers. Using this problem formulation, we evaluate heuristic solutions to this problem. We evaluate the performance of these solutions in stochastic load environments. We show that depending on the variance in the loads of the virtual machines being packed, some algorithms may outperform others.

3.1 Introduction

One of the major causes of energy inefficiency in data centers is the idle power wasted when servers run at low utilization [23]. In 2005, data centers accounted for 0.8% of all world energy consumption, costing $7.2 billion (US) [12]. Part of the problem is that most servers and desktops are in use only 5-15% of the time they are powered on, yet most x86 hardware consumes 60-90% of normal workload power even when idle [26, 4, 5].

Data center costs can be reduced by utilizing virtual machines (VMs). Using virtualization, multiple operating system instances can run on the same physical machine, exploiting hardware capabilities more fully, allowing administrators to save money on hardware and energy costs. To maximize the savings, administrators should assign as many VMs as possible to servers given performance requirements. We refer to this problem as the *Virtual Machine Assignment Problem*.

The Virtual Machine Assignment Problem (VMAP) is the problem of, given probabilistic distributions over the VM load, find an initial assignment which distributes the load on the VMs such that all have access to adequate resources and the number of servers used is minimized.

This type of problem might need to be solved at an e-commerce web site, which generally have times of lower hardware utilization. During these times of lower utilization, the web site acquires very few sales and therefore has more liberty to move VMs around. Once the day begins however, traffic will pick up and administrators will be less able to move VMs around. A good initial assignment means moving fewer VMs during peak hours of production. VMAP is not the problem of reassigning load after an initial assignment has already been made. We address this issue separately.

VMAP can be seen as a type of Bin Packing Problem, the problem of assigning a set of items into a set of bins, minimizing the number of bins in use [10]. However, VMAP is not as simple as the conventional Bin Packing Problem. VMAP is different in two important ways.

1. Each server has multiple types of constrained resources which the VMs consume. Each VM adds some amount of load to each resource type provided by the server, such as memory, disk space and CPU.
2. Loads that VMs exert on servers are probabilistic and not completely known ahead of time.

Prior research has partially addressed VMAP [21, 24]. One thing that prior research has not discussed is what happens when loads are probabilistically distributed. Specifically, we wish to investigate how load distributions can be incorporated into packing virtual machines onto servers. Prior research has not focused on this specific sub-aspect of VMAP.

This paper presents a novel way of taking expectations on loads of virtual machines. If system administrators have knowledge about the loads of virtual machines, expectations can be taken at different points in the probabilistic load distribution. The purpose of this paper is to present a way that this process can be done and to investigate some consequences of treating loads probabilistically.

We outline the paper here for reference. In Section 3.2, we describe our background research. In Section 3.3, we describe the model that we propose in this paper. In Section 3.4, we describe the assignment algorithms that we will compare. In Section 3.5, we discuss the metrics to determine success in our results. In Section 3.6, we detail our experimental setup. Finally, in Section 3.7, we discuss the results of our experiments.

3.2 Background Research

Different aspects of server consolidation have been studied and modeled [22, 20, 25, 1]. Other literature has focused on decreasing power use in virtualized environments [16]. In this section we review background research in three parts. First, prior research on modeling server load will be discussed. Second, We will discuss the Bin Packing problem. Third, we will briefly describe Genetic Algorithms as this is the foundation for one solution technique.

3.2.1 Virtual Machine Assignment

The problem that we identify in this paper as VMAP takes other names in literature. Stillwell et al. [24] define ResAlloc, which is a Mixed Integer Linear Program formulation of VMAP. In their problem formulation, they consider maximizing the yield, which represents the faction of a job's achievable compute rate that is achieved, on the server with the minimum yield. They then identify different solutions and evaluate the solutions on ResAlloc.

Song et al. [21] created RAINBOW, a prototype to evaluate a multi-tiered resource scheduling scheme on a workload scenario reflecting resource demands of services in a real enterprise environment. They first define resource flowing as the process in which resources released by some VMs/services are allocated to others. Their main contributions were a multi-tiered resource scheduling scheme for a VM-based data center, a model for resource flowing using optimization theory, and a global resource flowing algorithm.

Something that has not been investigated well in prior research is the fact that virtual machines are probabilistically distributed. We investigate what happens when we know something about the probabilistic nature of loads on virtual machines. This paper will give system administrators knowledge on what they should do, given that they have prior knowledge of how their virtual machines are distributed.

3.2.2 Conventional Bin Packing

In this paper we build a model that, given a set of VMs, each with a distribution of resource utilizations, decides how VMs should be assigned to servers. We present our model as a variant of the Conventional Bin Packing Problem. The Bin Packing Problem is the problem of finding the assignment of items to bins under which the number of bins is minimized.

Definition 7. The *Bin Packing Problem* is formulated as follows. Given a finite set of n items $I = \{1, 2, \ldots, n\}$ with corresponding weights $W = \{w_1, w_2, \ldots, w_n\}$ and a set of identical bins each with capacity $\mathcal{C}$, find the minimum number of bins into which the items can be placed without exceeding the bin capacity $\mathcal{C}$ of any bin. A solution to the Bin Packing

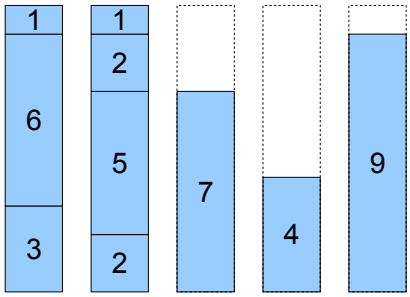

Figure 3.1: An inefficient Bin Packing solution.

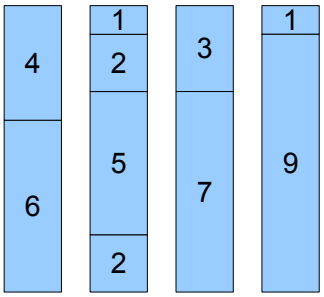

Figure 3.2: An efficient Bin Packing solution.

Problem is of the form $B = \{b_1, b_2, \ldots, b_m\}$, where each b_i is the set of items assigned to bin i, and is subject to the following constraints:

1. $\forall i \, \exists! \, j$ such that $i \in b_j$ (Every item belongs to one unique bin.)
2. $\forall j \, \sum_{n \in b_j} w_n \leq \mathcal{C}$ (The sum of the weights of items inside any bin cannot be greater than the bin capacity.)

In the Bin Packing Problem, the objective is to assign the set of items into a set of bins, minimizing the number of bins used. This idea is illustrated in Figures 3.1 and 3.2. Figure 3.2 shows an assignment that uses the same items as the assignment shown in Figure 3.1, but packs them in fewer bins.

Doing an initial placement of VMs onto servers can be seen as a type of Bin Packing Problem. In the Bin Packing Problem, a set of items are placed into bins, minimizing the number of bins. In the problem of placing VMs onto servers, VMs are assigned to servers, minimizing the number of servers, while assuring that some performance criteria is met.

Because these two problems are similar in this way, we will model the problem of making an initial assignment of items to servers as a type of Bin Packing Problem.

The Bin Packing Problem has been shown to be NP Hard [3]. We solve the problem that Bin Packing is inherently intractable by using approximation algorithms to solve the problem in our experiments.

One reason why the conventional Bin Packing Problem itself cannot be used to model the VM Assignment Problem is that there is no way in the conventional Bin Packing Problem to model multiple resources on one server. For that reason, we defer to prior research and model this problem using the Multi-Capacity Bin Packing Problem [29]. We will describe this problem in more detail in Section 3.2.3.

3.2.3 Multi-Capacity Bin Packing Problem

Definition 8. The *Multi-Capacity Bin Packing Problem* or *Vector Packing Problem* is similar to the conventional Bin Packing Problem that was given in Definition 7, but not identical. In the Multi-Capacity Bin Packing Problem, the capacity is a d-dimensional vector $\mathcal{C} = \langle C_1, C_2, \ldots, C_d \rangle$ where d is the number of resource types. The weights are redefined so that the weight of item i is a d-dimensional vector $\boldsymbol{w}_i = \langle w_{i_1}, w_{i_2}, \ldots, w_{i_d} \rangle$ [24, 13].

1. $\forall i \, \exists! \, j$ such that $i \in b_j$ (Every item has to belong to some unique bin.)
2. $\forall j \, \forall k \, \sum_{n_k \in b_j} w_{n_k} \leq C_k$ (The sum of all the weights for any capacity for any bin must be less than the corresponding capacity for that bin.)

Note that in the Multi-Capacity Bin Packing Problem the resources consumed by each VM accumulate, up to some maximum. Items are placed on each corner to show that the sum of the weights for any one type has to be less than the corresponding bin capacity. The weights on items in the same bin are additive. For all resources, the sum of all weights on items of that particular resource must be less than the corresponding resource capacity on the server.In VMAP, resources used by one VM can not be used by any VM on the same physical hardware.

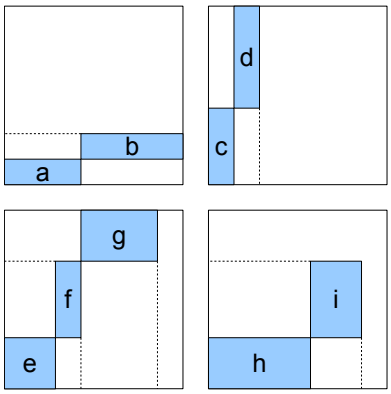

Figure 3.3: An inefficient Multi-Capacity Bin Packing solution.

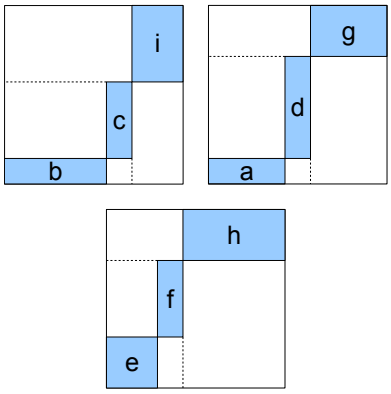

Figure 3.4: An efficient Multi-Capacity Bin Packing solution.

As with the conventional Bin Packing Problem, the objective is to minimize the number of bins. Figures 3.3 and 3.4 illustrate this principle. Figure 3.4 packs the exact same items as are found in Figure 3.3 in three bins instead of four. This means one fewer server drawing power.

The Multi-Capacity Bin Packing Problem lends itself better to the problem of assigning VMs onto servers than the conventional Bin Packing Problem. In the conventional Bin Packing Problem, there is no way to model the multiple different resources that are available on a server, such as CPU, RAM and disk bandwidth. Because the Multi-Capacity Bin Packing Problem lends itself well to modeling the many different resources of a server, we use this problem formulation in our optimization techniques.

3.2.4 Genetic Algorithms

Bin Packing Problems have been solved with Genetic Algorithms (GAs). There is no rigorous definition for GAs [15]; they derive much of their inspiration from Darwinian biological processes. In GAs, individuals represent candidate solutions to the problem. These candidate solutions explore the solution space by undergoing processes similar to those of biological organisms. The simplest form of genetic algorithm involves three types of operators:

- **Selection**–Individuals in the population are selected for crossover with other individuals. Usually, selection is based on elitism, where the more fit individuals are selected more often than less fit individuals.
- **Crossover**–Two individuals in the population exchange subsections of their candidate solution with each other to create new offspring.
- **Mutation**–After crossover, each individual has a probability of having their candidate solution modified slightly.

3.3 Description of Model

As described in Section 3.2, we model VMAP by using the Bin Packing Problem. However, there are a few differences between the Bin Packing Problem and VMAP which were identified in Section 3.1. The model that we will use for VMAP is based on the Multi-Capacity Bin Packing Problem, as discussed in Section 3.2.3, and it also uses probabilistic estimates of loads, which we will discuss here.

3.3.1 Probabilistic Estimates

The second way that VMAP is different from the conventional Bin Packing Problem is that the loads VMs exhibit on servers are not known completely when initial assignments are made. Even though these loads are not completely known ahead of time, probabilistic estimates can be made for loads on VMs. Therefore, we treat loads as probabilistic. There are at least two ways in which system administrators can derive probabilistic estimates for loads on VMs.

1. If the system administrators have reason to believe that VM loads can be characterized as a type of known probabilistic distribution, this problem becomes the problem of parameter estimation. Using data, it is possible to estimate the parameters of a parametric probabilistic model using known methods such at methods of moment or maximum likelihood estimation.
2. If system administrators do not know the probabilistic distribution which describes the load on VMs, a *nonparametric model* can be used. We will describe nonparametric distributions in more detail in Section 3.6.

We assume that the probabilistic distribution on the future load for each resource for each VM is to the algorithm ahead of time. Recall as well that in the Multi-Capacity Bin Packing Problem, weights are deterministic and known instead of probabilistic and unknown. This means that if the Multi-Capacity Bin Packing Problem is to be used as a model, some estimate or expectation of the probabilistic distribution must be given to the algorithm

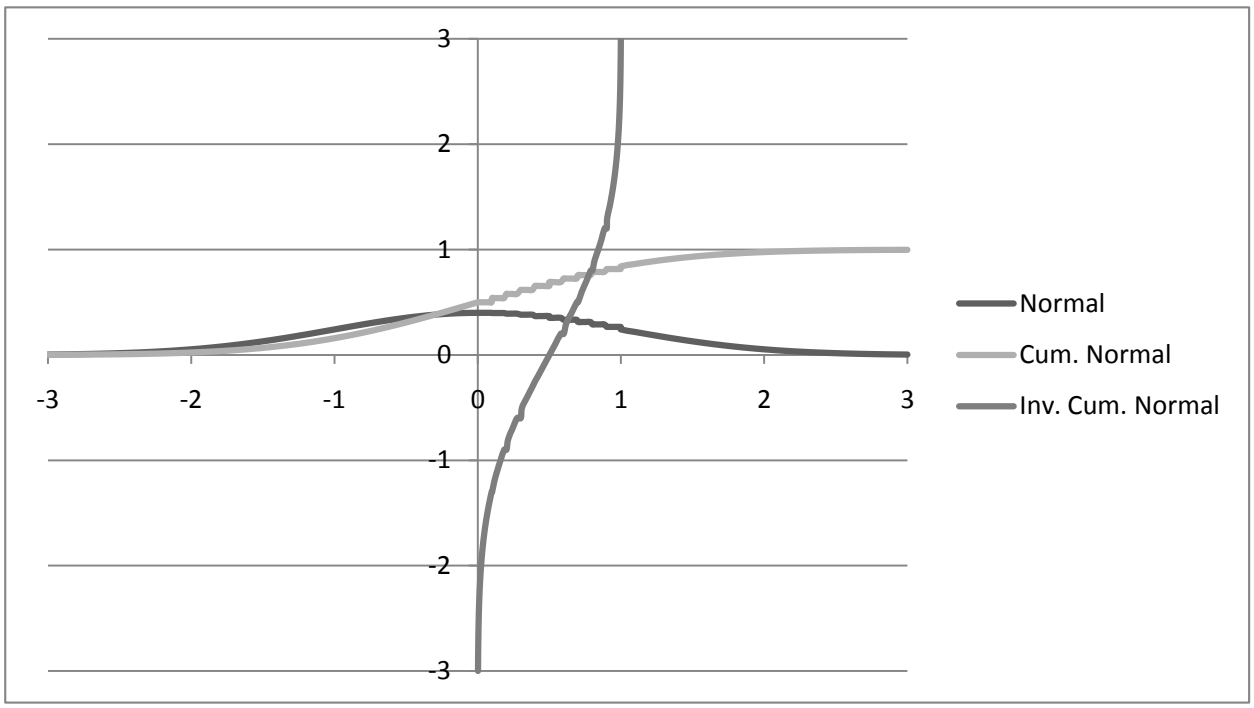

Figure 3.5: The pdf, cdf and icdf for the normal distribution with mean $\mu = 0$ and variance $\sigma^2 = 1$.

solving VMAP. In this section, we will explore how to make this estimate from probabilistic distributions of the load.

Recall that the *probability density function (pdf)* $f(X)$ of a random variable X describes the relative likelihood of X to occur at a given point in the observation space. Recall, also, that the *cumulative distribution function (cdf)* $F(X)$ of a random variable X is defined for a number χ by:

$$F(\chi) = P(X \leq \chi) = \int_{-\infty}^{\chi} f(s)ds \tag{3.1}$$

where $f(s)$ is the likelihood associated with the random variable X obtained from the pdf f at s. The pdf for the normal distribution with mean $\mu = 0$ and variance $\sigma^2 = 1$ is shown in Figure 3.5.

The cdf of the normal distribution the same distribution with mean $\mu = 0$ and variance $\sigma^2 = 1$ is also shown in Figure 3.5. The cdf $P(X \leq \chi)$ is the probability that X

is less than or equal to χ. It answers the question of "What is the likelihood of getting any load less than a specified load for a particular resource for a VM?" However, even though this metric may be useful, a better question to ask may be in the opposite order. "What is the maximum load y for a given likelihood z, such that $F(y) \leq z$?" This question can be answered by using the inverse cumulative distribution function.

The *inverse cumulative distribution function* (icdf) or *quantile function* returns the value below which random draws from the given cumulative distribution function would fall, $p * 100$ percent of the time. That is, it returns the value of χ such that

$$F(\chi) = Pr(X \leq \chi) = p \tag{3.2}$$

for a given probability $0 < p < 1$.

Using the icdf, we can specify a percentile value and obtain a corresponding load which can be passed to the assignment algorithm. Using the value from a high percentile will result in a high load being passed to the assignment algorithm, and tend to make the assignment more robust to random variation. Lower percentiles result in assignments more likely to become over loaded. Using the quantile function to decide which load to use means that the algorithm designer can incorporate any level of robustness or aggressiveness into the algorithm. Figure 3.5 shows the inverse cumulative distribution function for the normal distribution with mean $\mu = 0$ and variance $\sigma^2 = 1$. We call the load derived from the icdf value the *derived load*. The derived load is the load used by the bin packing algrorithm when a concrete value must be used.

3.4 Initial Assignment Algorithms

Many companies and organizations do not use a structured approach to the initial placement of VMs onto servers. Even though a system administrator may view a summarization of the load for each VM, and place VMs onto servers using that summarization, we have not

found any formalization of a model for VMAP, nor any algorithm meant to make an initial placement of VMs to servers. Because this is a new model, and the model derives its roots in the Bin Packing Problem, we present some well-known algorithms that solve the Bin Packing Problem for consideration. We will describe and compare four assignment algorithms in this paper–Worst Fit, First Fit, Permutation Pack [13] and Reordering Grouping Genetic Algorithm [29].

3.4.1 Worst Fit

The worst fit algorithm for bin packing considers each item in order. For each item, first, it considers only bins that already have at least one item placed in them. It places the item in the bin into which it fits which has the most amount of free space. If the item does not fit in any of the bins considered, it is placed into a new bin.

The worst fit algorithm is considered in this paper because of its tendency to leave a bit of space in every bin that it uses. This extra space may be helpful when observed loads on VMs exceed what the assignment algorithm expected.

Finding the emptiest bin is easy to do in the conventional Bin Packing Problem as each item only has one weight. The sum of the weights of all the items in any one bin can be used to give a number describing the fullness of a bin. However, finding the emptiest server in VMAP is not as obvious because each VM has multiple loads which it exerts on the server. In order to compare the emptiness of one server to another's, there must be a way to combine the different loads. We use a Euclidean distance metric to compare the amount of free space in two different bins.

3.4.2 First Fit

The first fit algorithm for bin packing is a way to place an initial set of VMs on servers. In the first fit packing algorithm, items are arranged in order (often decreasing order). Bins are

also arranged in a list. Each item is then considered in order and placed into the first bin into which it fits.

3.4.3 Permutation Pack

Permutation Pack (PP) attempts to find items in which the largest w components are exactly ordered with respect to the ordering of the corresponding smallest elements in the current bin [13]. Let R_i denote the remaining space in the ith capacity of a particular bin. If $d = 2$ and $R_1 < R_2$, then we look for an item n such that $w_{n_1} < w_{n_2}$. If no item is found, the requirements are continually relaxed until one is found. One of the weaknesses of Permutation Pack is the running time. If all permutations are considered, it runs in $O(d!\, n^2)$ where d is the number of resource types and n is the number of items to be packed. We refer the reader to Leinberger et al. [13] for further description of the algorithm.

3.4.4 Reordering Grouping Genetic Algorithm

Reordering Grouping Genetic Algorithm (RGGA) is a genetic algorithm that was developed for the Multi-Capacity Bin Packing Problem [29]. RGGA has been shown to solve the Multi-Capacity Bin Packing Problem quickly, developing good solutions. RGGA represents an instance of the bin packing problem not only as an assignment of items to bins, but also as a list of items to be first fit packed. RGGA's crossover operator is an exon shuffling crossover as defined by Rohlfshagen et al [18]. RGGA uses a mutation operator that swaps two items in the first fit list $\frac{1}{3}$ of the time, moves an item from one spot to another in the first fit list $\frac{1}{3}$ of the time, and eliminates a bin, reinserting the contents $\frac{1}{3}$ of the time. This last idea is the mutation operator used by Faulkenauer in his Grouping Genetic Algorithm [7].

RGGA was shown to find optimal solutions to the bin packing algorithm in fewer iterations than the leading genetic algorithms in literature. As well, RGGA was shown to generate very good solutions to even large problem sizes of the Multi-Capacity Bin Packing Problem.

3.5 Metrics to Determine Success

In this section, we investigate two different metrics to determine the level of success of an initial VM assignment. These two metrics are total number of servers used and the proportion of servers over capacity.

3.5.1 Number of Servers Used

The total *number of servers* used is defined as the number of servers upon which VMs are placed. This metric is useful in determining the tightness of a particular assignment. An assignment which places its VMs on fewer servers will likely save energy in the long run. Aggressive assignment algorithms often maximize this metric.

3.5.2 Proportion of Server Resources Over Capacity

A server resource is over capacity if VMs on the server request more of that resource than is available on the server. For example, if the sum of the total amount of RAM requested by all the VMs on a particular server is greater than the amount available on the server, then the RAM on the server would be considered over capacity. In order to calculate the proportion of server resources over capacity, we divide the sum of all total server resources over capacity by the sum of all total server resources. The algorithm for calculating the *proportion of servers over capacity* is shown in Equation 3.3.

$$\frac{\sum_{b_i \in B} \sum_{j \in b_i} F(\sum_{w_{j_k} \in \boldsymbol{w}_k} w_{j_k}, C_k)}{\sum_{b_i \in B} \sum_{j \in b_i} 1} \tag{3.3}$$

where $F(X, Y)$ returns 1 if X is greater than Y and 0 otherwise.

Conservative VM assignment algorithms perform worse with regard to this metric, because they, on average, have less allocated resources per server. If a particular VM uses more server resources than was allotted to it, the server might not be over capacity if the

algorithm chose to leave extra room. Algorithms that are more conservative when assigning VMs also yield solutions with a greater total number of servers.

3.6 Experimental Setup

Because the contribution of this paper is the model proposed for VMAP, we wish to validate this model in our results by showing that the modifications we made to the conventional Bin Packing Problem are indeed helpful in modeling VMAP.

In our experiments, we did exactly what we expect real system administrators to do with our work. We deployed various VMs to a cluster of computers, gathered data on resource utilization of these VMs, generated an assignment for these VMs to servers, and carried out that assignment with virtualization software. The VMs that we created were of the form such that 8-12 of them would fit on a physical server. Because deployments in real data centers normally have 8-12 VMs per physical server [6], we expect our results to generalize well to other clusters.

We use the data itself as a nonparametric statistical distribution. The mean of this distribution can be computed by finding the mean of the data. The variance of the distribution can be found by finding the variance of the data. This distribution can be sampled by picking a data point with uniform probability. The icdf of this distribution can be found for any percentile by sorting the data, multiplying the percentile used by the number of data points, and returning that particular data point.

With this nonparametric distribution for the load on each VM, we were able to generate new assignments of VMs to servers. We ran each assignment algorithm for varying icdf values. We show the predicted performance for varying icdf values in our results. After simulating the assignment of VMs to servers, we predicted the number of servers and proportion of servers over capacityfor the case if we actually carried out the assignment. In order to obtain our simulated metrics, we sampled from the load distribution for each VM

and assigned loads to VMs from their distributions. Using this data, we obtained predicted results for what a assignment would be like if we carried out the assigning on the servers. [1].

After obtaining predicted results for assigning VMs to servers, we then reassigned VMs to servers, averaged the results and analyzed how closely our model showed what happened in real life. We show the results of these new assignments in our results. We used the Kernel-Based Virtual Machine (KVM) kernel virtualization infrastructure, the qemu processor emulator, and the libvirt virtualization management tool. In our data set, we used VMs with varying CPU and RAM loads. We kept track of the loads on the VMs and recorded the observed CPU and RAM utilization on host servers.

Because it is an option that the system administrator can tweak, we also show the predicted performance for different algorithms for varying icdf values. As mentioned in Section 3.4, raising the icdf value makes an assignment algorithm more conservative and lowering the icdf value makes an assignment algorithm more aggressive.

In our simulated experiments, we performed these steps:

1. The packing algorithm receives some type of distribution over the load for each virtual machine it needs to pack.
2. The packing algorithm derives a specific load using the distribution from step 1 and the icdf value used.
3. The packing algorithm develops an assignment of virtual machines to servers.
4. One specific load for each virtual machine is sampled from the load distribution for that virtual machine. This load is assigned to that virtual machine.
5. Metrics are gathered.
6. Steps 1-4 are repeated as needed.

When we implemented RGGA, we used a maximum function evaluation count of 7500, a crossover rate of 0.8, and a mutation rate of 0.1. In the event that two individuals

[1] `http://aml.cs.byu.edu/~davidw/virt/datasets`

do not crossover, one of them is picked randomly and added directly to the next population. If an individual is not mutated, it is simply added as is to the next generation.

3.7 Results

In our results, we wish to validate the probabilistic model proposed in Section 3.3 and the algorithms we proposed in Section 3.4. We will divide presentation of results in three parts. First, we will show how system administrators can use assignment algorithms with varying icdf values. We then analyze and present graphs that show in a practical standpoint number of servers used and the proportion of servers over capacity. Lastly, we analyze how closely our predictive model resembles what happens on real hardware by repacking VMs to servers.

Even though we do not directly incorporate probabilistic loads into any assignment algorithm proposed in this paper, we show results that help system administrators to indirectly incorporate probabilistic results into the assignment algorithm picked by applying the ideas presented in Section 3.3.1. We systematically increased the icdf value and subsequently the derived load from the icdf value. Increasing this value increases the derived loads on VMs which the algorithm uses. As discussed earlier, lower values of percentile yield more conservative assignment algorithms and higher value of percentile yield more aggressive assignment algorithms.

Figure 3.6 shows the proportion of servers over capacity while Figure 3.7 shows the number of servers found by the different algorithms. The independent variable in both graphs is the icdf value used. Even though we present both figures separately, they must be interpretted together. One shows the number of servers used, while the other shows the proportion of servers over capacity. Algorithms which tend to use fewer bins will tend to look better in Figure 3.7, but look worse in Figure 3.6.

The icdf value is merely a parameter used to determine both the number of servers used and the proportion of servers over capacity. The icdf value used is really irrelevant because the icdf value picked by system administrators is a function of the number of servers

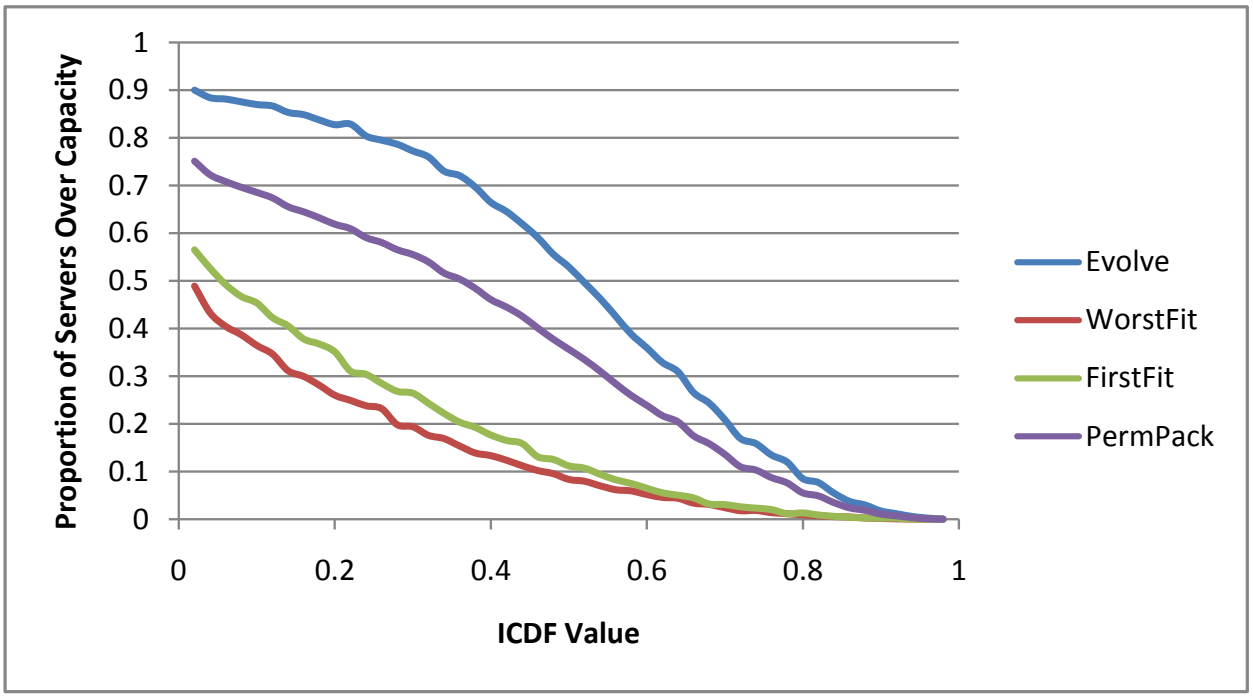

Figure 3.6: A comparison of the proportion of servers over capacity for varying percentile values.

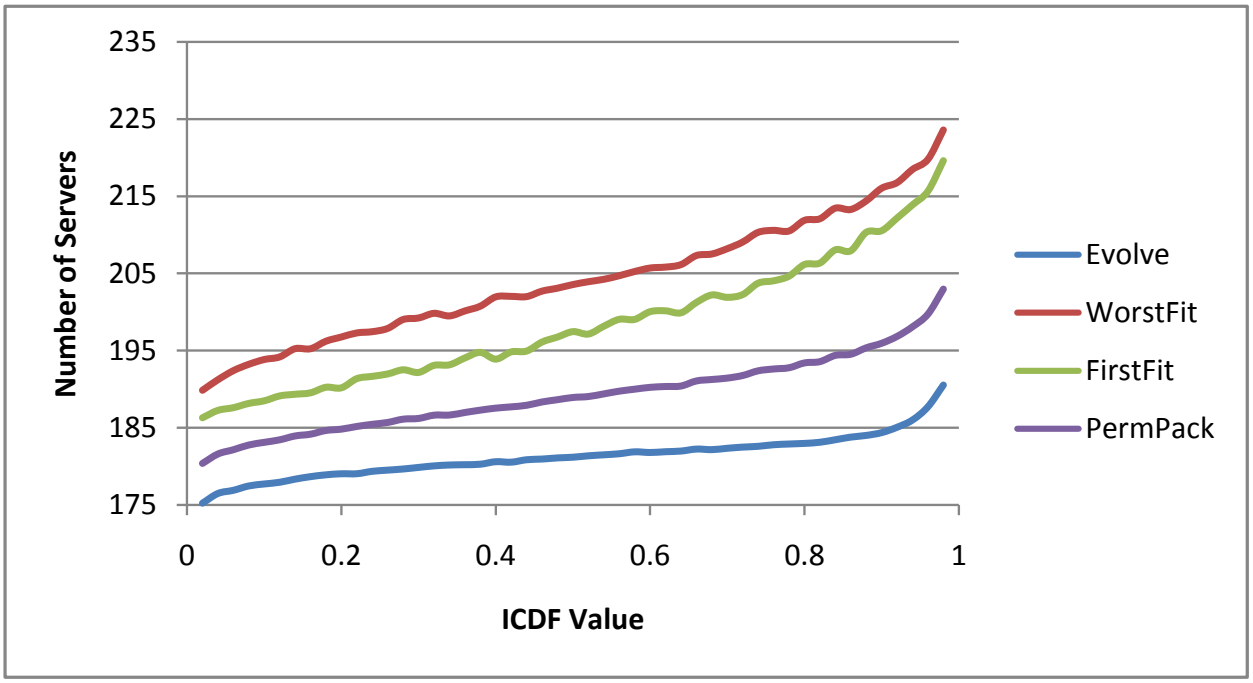

Figure 3.7: A comparison of the the number of servers found for varying percentile values.

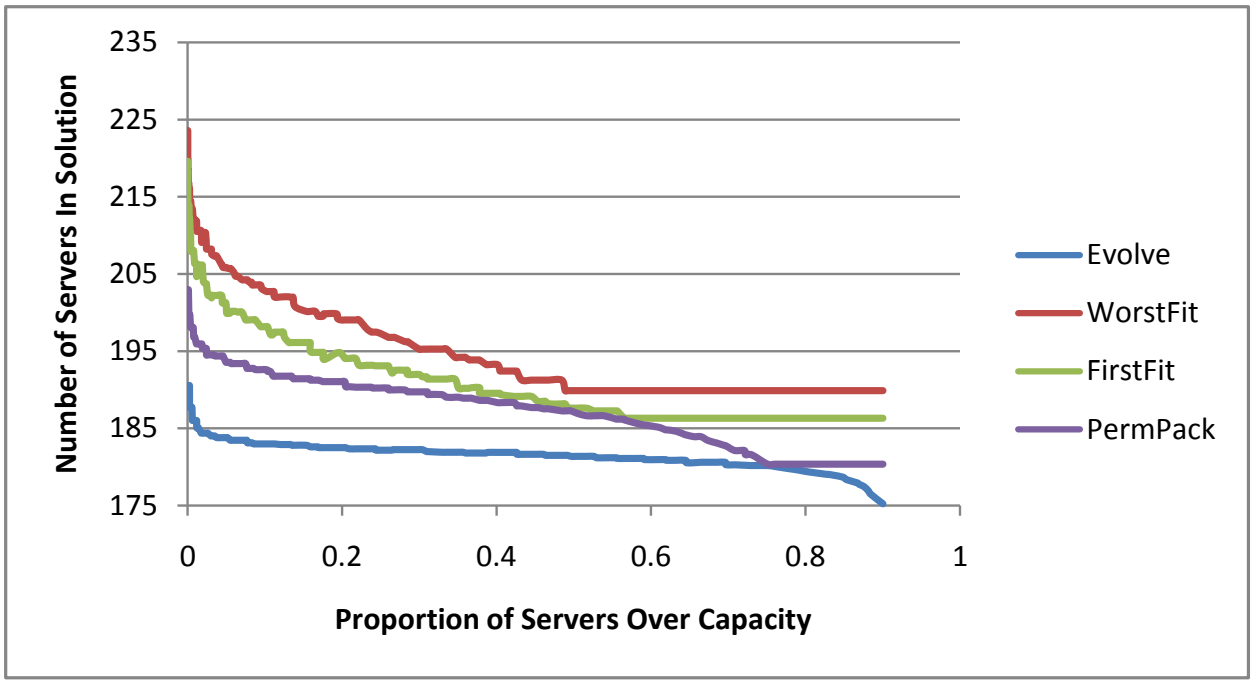

Figure 3.8: A comparison of the proportion of servers over capacity with the number of servers found.

used and the proportion of servers over capacity. Instead of showing this graph, we wish to combine Figures 3.6 and 3.7 so that we can see this type of interaction between the proportion of servers over capacity and the number of servers used.

In order to simplify our analysis, we joined the proportion of servers over capacity with the number of servers found using the percentile values to produce a joint graph. Using this graph, Figure 3.8, a system administrator can choose how many servers on average will be in utilization or what proportion over capacity they are willing to tolerate in order to get the other parameter. This graph shows that for this particular configuration, RGGA

3.7.1 Repacking to Servers

Lastly, in order to validate the model we generated, we used the assignments generated to make assignments of real VMs to servers. We measured the proportion of server resources over capacity for all three assignment algorithms at the 80th cdf percentile. We show our results in Figure 3.9. The predicted proportion of server resources over capacity all algo-

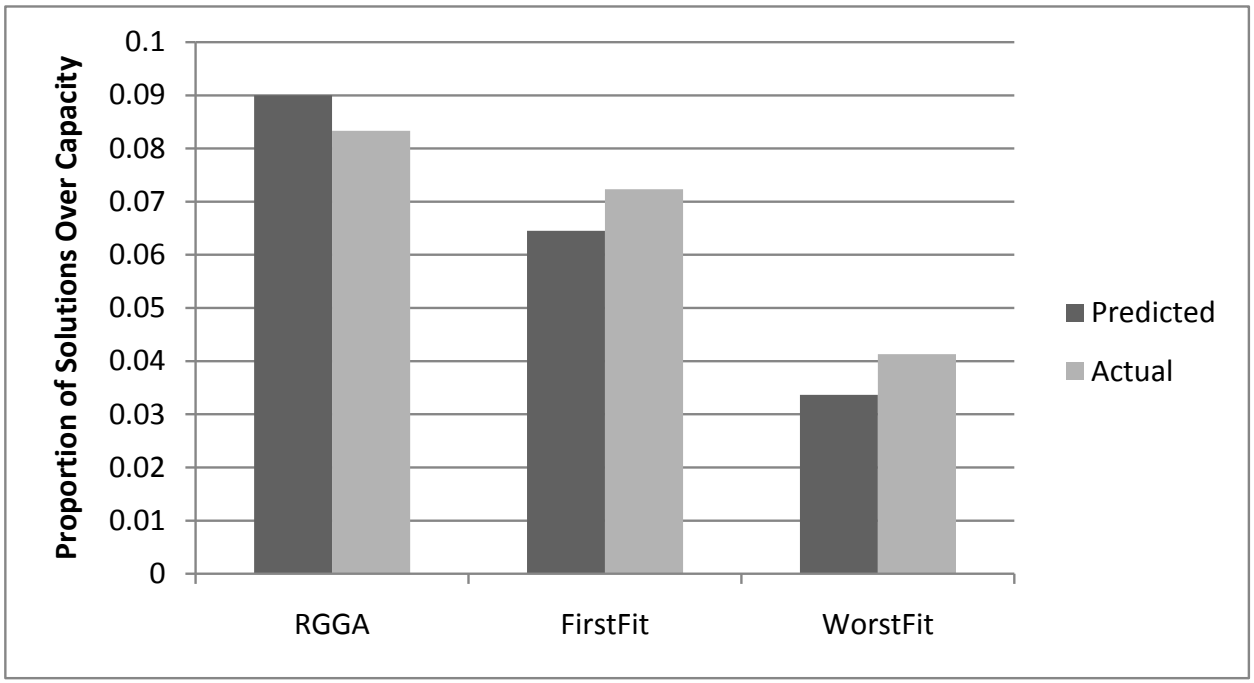

Figure 3.9: A comparison of the proportion of solutions over capacity that our algorithm predicted and also the measured proportion of solutions over capacity when deploying real VMs to servers for the 80th cdf percentile.

rithms is very close to the actual proportion of server resources over capacity measured when deploying VMs to servers. This gives validity to the model we suggest in this paper.

3.8 Conclusion

A novel model for Virtual Machine Assignment Problem is proposed. This model uses ideas from the conventional Bin Packing Problem, where servers are bins and VMs are items, with two variations. First, it allows multiple weights for each item and multiple capacities for each bin. The sum of all the weight of any one type in any bin must be less than that corresponding capacity in that bin. Second, our model proposes that VMs have probabilistic loads. The probabilistic loads should be incorporated into the assignment algorithm for best results. We, show the feasibility of using probabilistic loads with the assignment algorithm by modifying three known packing algorithms.

Adding probability theory helps system administrators to pick the correct percentile value representing the spot on the inverse cumulative distribution function representing the load of a resource. Small values for the icdf value yield more conservative assignment algorithms while larger values for the icdf value yield more aggressive assignment algorithms.

For the problems which we investigated, it seems that optimization algorithms like RGGA perform well.

Chapter 4

An Analysis of Variance in Virtual Machine Packing

We formally define virtual machine variance and show its relevance in the Virtual Machine Packing Problem. We show that system administrators can use virtual machine variance as a tool for deciding which packing algorithm to use. We systematically increase virtual machine variance to see its effects on different packing algorithms. We show that for packing problems where virtual machines have low variance, Reordering Grouping Genetic Algorithm performs very well, while for problems where virtual machines have high variance, all algorithms compared perform equally. We then test hybrid problems where some virtual machines have low variance, and some have high variance. We show that Reordering Grouping Genetic Algorithm performs well on these types of problems.

4.1 Introduction

One of the major causes of energy inefficiency in data centers is the idle power wasted when servers in data centers run at low utilization [23]. In 2005, data centers accounted for 0.8% of all world energy consumption, costing $7.2 billion [12]. Part of the problem is that most servers and desktops are in use only 5-15% of the time they are powered on, yet most x86 hardware consumes 60-90% of normal workload power even when idle [4, 5, 26].

Data center costs can be reduced by utilizing virtual machines (VMs). Using virtualization, multiple operating system instances can be placed on the same physical machine, exploiting hardware capabilities more fully and allowing system administrators to save money on hardware and energy costs. To maximize the savings, system administrators should place as many virtual machines as possible onto a server while satisfying performance criteria. This problem is referred to as the *Virtual Machine Assignment Problem* [28].

The Virtual Machine Assignment Problem (VMAP) is the problem of given virtual machine loads with no packing before-hand, find an initial packing which distributes the load on the virtual machines as well as possible, minimizing the number of servers used.

In defining VMAP, we proposed a model for virtual machine assignment where virtual machines all have probabilistic loads [28]. In this model, loads on virtual machines can be represented using probability distributions. Each load has a mean which represents the expected value of all observed loads along with some variance that represents the amount of dispersion of a given load.

The decision of how to model load distributions of virtual machines is an important part of modeling VMAP. One part of modeling these distributions is the amount of variance attributed to the distribution. The variance of the load distribution is the amount of uncertainty a system administrator has about the future load. It is possible that virtual machines with less variance should be packed differently than virtual machines with more variance.

In our previous paper defining VMAP, we did not investigate fully the implications of what happens if system administrators know the variance of virtual machine loads. In this

chapter, we show that if system administrators are rather certain about the loads of virtual machines (meaning that the virtual machines have low variance), then Reordering Grouping Genetic Algorithm (RGGA) can be used and outperforms other heuristics [29].

4.2 Experimental Setup

In our experiments, we measured two different types of load on virtual machines–RAM and CPU usage. We assumed that loads on the virtual machines were normally distributed. Using the normal distribution, variance can be encoded directly as a model parameter.

The experiments performed in this section are similar to those we performed when defining VMAP. For each of these three algorithms, we varied the inverse cumulative distribution function number that each one used. Each algorithm observed a load based on the inverse cumulative distribution function number used as was done in Chapter 3.

When conducting my tests, we performed a variety of steps:

1. The packing algorithm receives some type of distribution over the load for each virtual machine it needs to pack.
2. The packing algorithm develops an assignment of virtual machines to servers.
3. One specific load for each virtual machine is sampled from the load distribution for that virtual machine. This load is assigned to that virtual machine.
4. Metrics are gathered.
5. Steps 1-4 are repeated fifty times in order to achieve statistical significance for the test.

In our results, we found the average number of servers and average percent of servers over capacity for each algorithm and inverse cumulative distribution function value.

4.3 Results

We divide our results into two sections. In the first section, we analyze what happens when virtual machines all have equal variance and this variance is systematically increased. We also observe what happens for extremely low and high variance values.

In the second section of our results, we analyze what happens when virtual machines have different variances. Specifically, we created two types of virtual machines. The first have low variance and the second have high variance. We wished to analyze what happens when these two types of virtual machines interact with each other.

For the purposes of our experiments, we use the same packing algorithms proposed in Section 3.4. Specifically, these are First Fit, Worst Fit and RGGA.

4.3.1 Virtual Machines Have Equal Variance

There are two variables which explain the type of packing developed. The first is the algorithm itself. Some algorithms tend to create looser packings than other algorithms. For example, the Worst Fit algorithm tends to spread load and out and creates a looser packing than RGGA.

The other variable which explains the type of packing developed is the icdf value used for the algorithm. Worst fit algorithm with a very low icdf value may produce a packing with fewer bins than RGGA with a very high icdf value.

In our first experiment, we incrementally changed the variance of virtual machines. Different packing algorithms assigned virtual machines using varying icdf values. Using the number of servers found and the percent of servers over capacity for each inverse cumulative function distribution value, we found which algorithm outperformed the others.

Our first observation was that for very low variance problems, RGGA outperforms other methods. Figure 4.1 shows the number of servers found for the three algorithms used for comparison. As can be seen, RGGA outperforms both of the other two algorithms. Intuitively, this finding makes sense; If the variance is zero for all loads on all virtual machines,

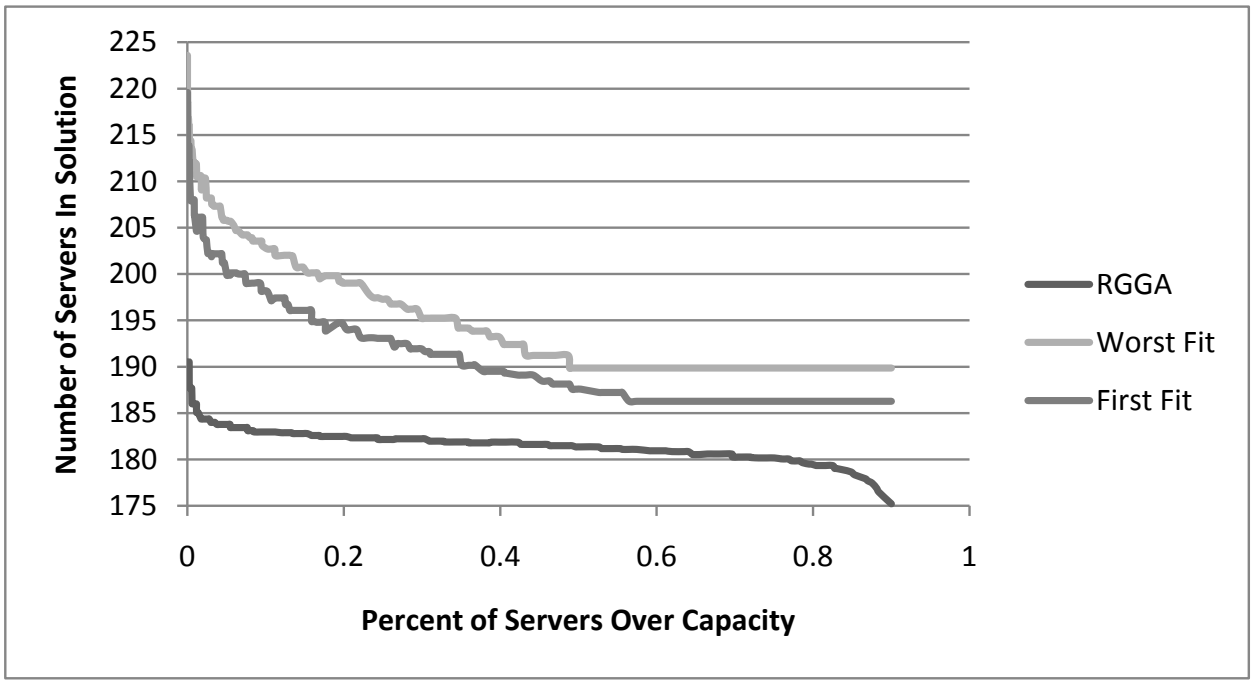

Figure 4.1: The number of servers found for varying algorithms for varying percents of servers over capacity for a problem with very low variance.

then there are no repercussions for packing virtual machines very tightly onto servers. Since RGGA means to pack VMs tighter onto servers than other methods, then it benefits from all the gains of packing tightly without suffering any of the repercussions.In low-variance problems, if actual loads consume more resources than were allotted to those virtual machines during the initial packing process, the unallocated resources naturally found on each server are usually enough to prevent any over capacities.

After finding that optimization pays off for low-variance problems, we performed experiments to discover what happens for high-variance problems. We show the results of these experiments in Figure 4.2. As can be seen in Figure 4.2, all algorithms performed similarly for high variance problems. This means that RGGA did not outperform the other algorithms for high variance problems. We saw these types of results in our paper that defined VMAP as well. There, we saw that for some high variance problems, all algorithms performed similarly. However, if system administrators know that the load on virtual machines will not

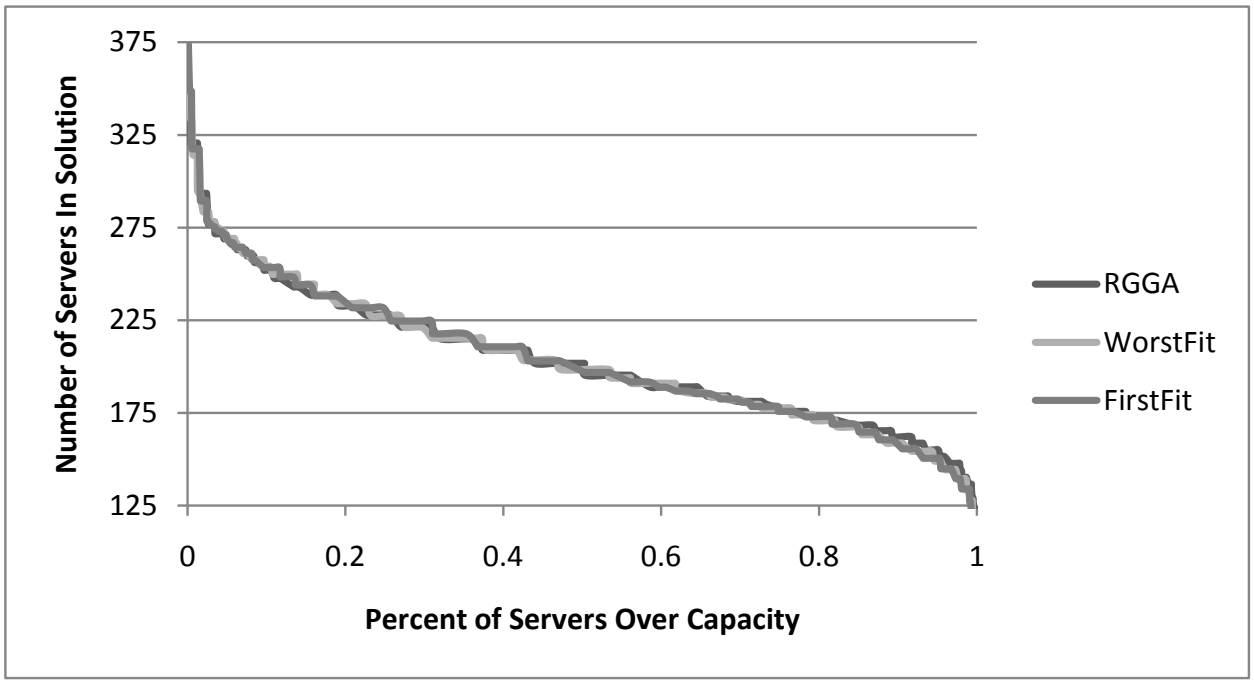

Figure 4.2: The number of servers found for varying algorithms for varying percents of servers over capacity for a problem with high variance.

fluctuate much, it makes sense that we should be able to exploit that fact and pack virtual machines tighter.

We started performing experiments systematically increasing the variance of virtual machines in order to find this point.

In Figure 4.3 we show the number of servers found plotted against the percent of servers over capacity for problems where loads have a standard deviation of about 1.7% of the total server capacity. In Figure 4.4 we show the number of servers found plotted against the percent of servers over capacity for problems where loads have a standard deviation of about 2.5% the total server capacity. Lastly, in Figure 4.5, we again show the number of servers found plotted against the percent of servers over capacity when loads have a standard deviation of about 3.1% the total server capacity.

Figure 4.3 shows that it is feasible to run RGGA to pack virtual machines tightly when virtual machines have load standard deviations of about 1.7% of the total server load. However, by the time the load standard deviation increases to 3.1% as shown in Figure 4.5,

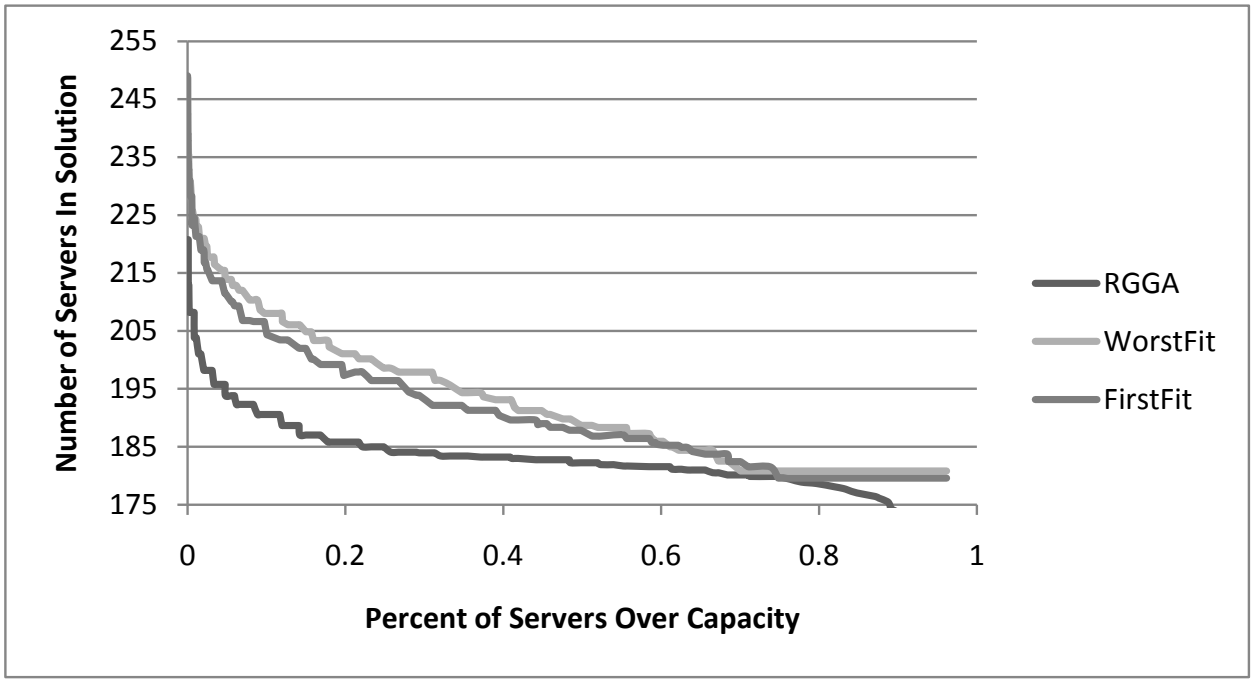

Figure 4.3: The number of servers found for varying algorithms vs percent of servers over capacity. Variance on VMs is set to 1.7% the total server capacity.

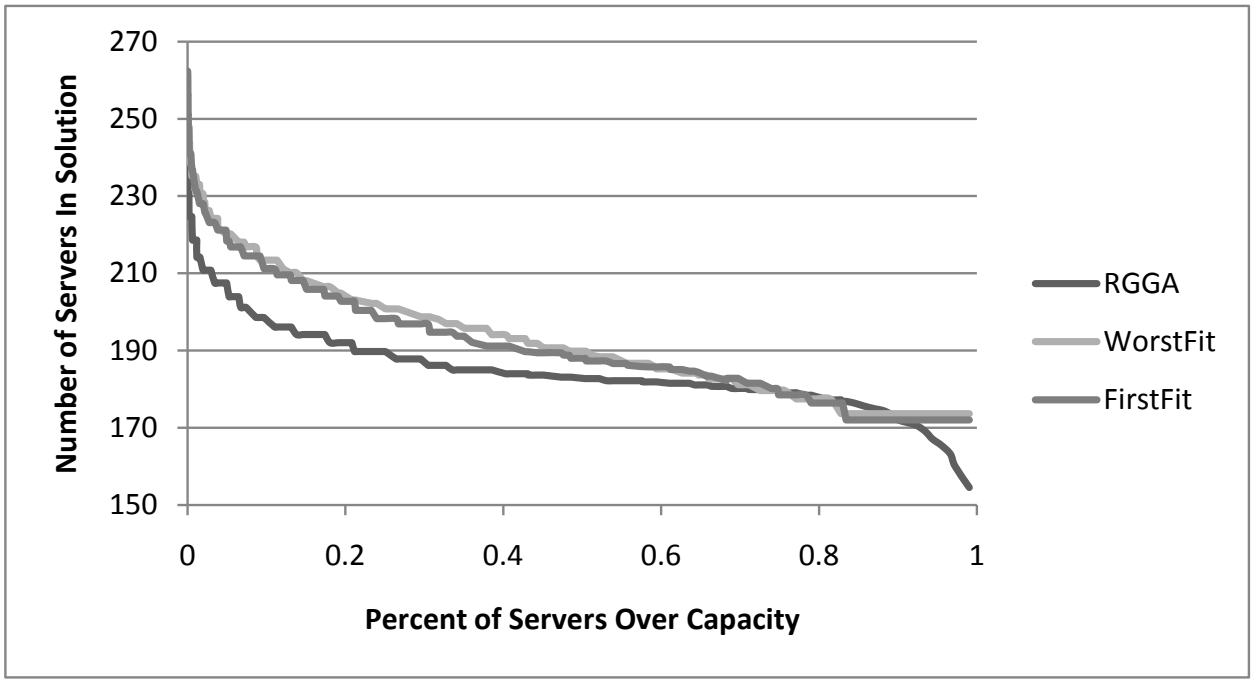

Figure 4.4: The number of servers found for varying algorithms vs percent of servers over capacity. Variance on VMs is set to 2.5% the total server capacity.

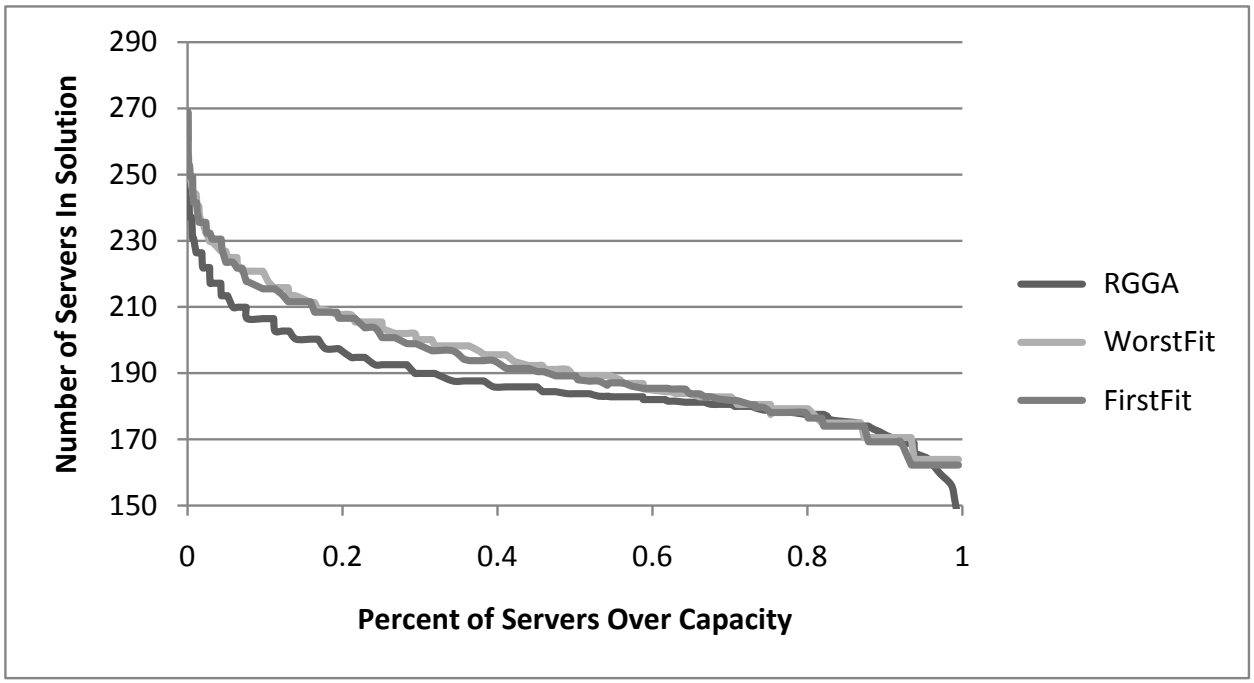

Figure 4.5: The number of servers found for varying algorithms vs percent of servers over capacity. Variance on VMs is set to 3.1% the total server capacity.

the gains from packing virtual machines tightly with RGGA can no longer be seen clearly. Therefore, for this problem type, for deployments where the loads on virtual machines have a standard deviation of less than 2-3 percent of the total server capacity, RGGA can be used to effectively pack virtual machines tightly. However, for this problem type, if loads on virtual machines have a standard deviation of more than 2-3 percent of the total server capacity, it seems that all packing algorithms considered here perform similarly.

4.3.2 Virtual Machines Have Differing Variance

In our second experiment, we had two types of virtual machines packed together. The first type of virtual machines could be packed efficiently and had a load standard deviation of 1.7% of the total capacity for the server. The second type of virtual machines could not be packed efficiently by an optimization algorithm. The standard deviation of the load of these virtual machines was 5.5% of the total capacity for the server. Fifty percent of the virtual

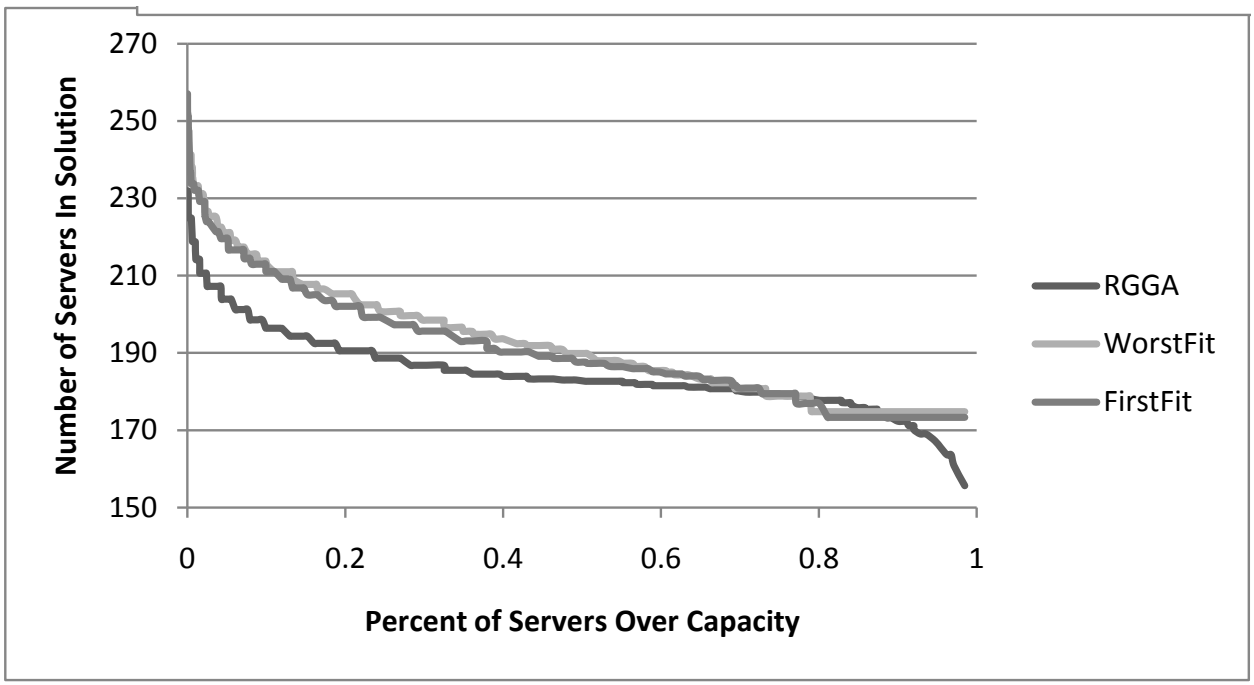

Figure 4.6: The number of servers found for varying algorithms vs percent of servers over capacity. Half of VMs have high variance. Half of VMs have low variance.

machines in this experiment were from each group. The average standard deviation of the load for virtual machines was 3.6% of the total server capacity.

Before performing this experiment, our hypothesis was initially that if a large number of virtual machines with high-variance loads are to be packed along with virtual machines with low-variance loads, then the virtual machines with low-variance loads would be overpowered by the virtual machines with high-variance loads. The virtual machines with high-variance loads would cause most servers to go over capacity. Our hypothesis was that RGGA would see poor performance for this type of problem.

We show in Figure 4.6 the resulting graph of having just a some virtual machines that could be packed together tightly, and others that had too much variance to be packed tightly. RGGA still outperforms the other algorithms. This means that our initial hypothesis that the high-variance loads would make all servers into which they are placed go overcapacity was wrong. It seems that what is most important is the combined variance of virtual machines in a server.

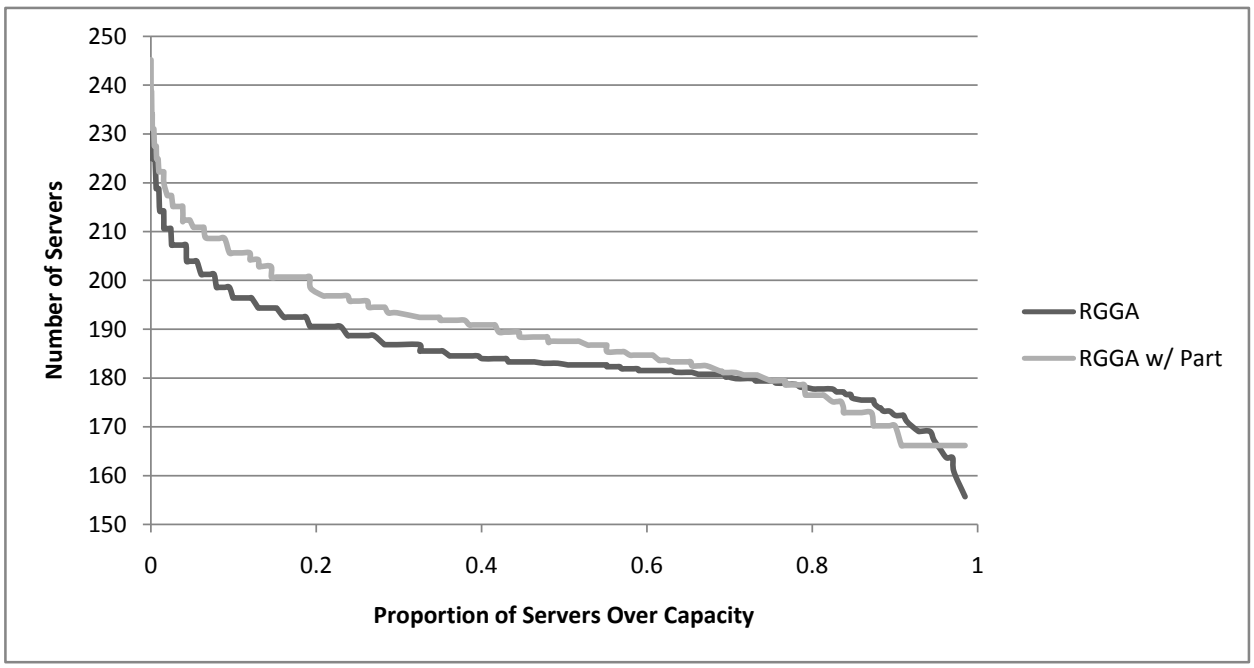

Figure 4.7: The number of servers found for RGGA and RGGA w/ Part vs percent of servers over capacity. Half of VMs have high variance. Half of VMs have low variance.

The results here led us to experiment with a new variant on RGGA. Our new variant (RGGA w/ Part) picks all the virtual machines with low variance and runs RGGA on those, hoping to pack them tightly together. Afterward, the algorithm can pack the high-variance virtual machines together, since those will probably go over capacity anyway. The motivation for trying this approach was to see if any gains could be made from keeping high-variance virtual machines away from low-variance servers. This way, the high-variance virtual machines would not make the servers where low-variance virtual machines are packed go over capacity.

We show the results of this experiment in Figure 4.7. The new method does not significantly outperform the existing method, and in fact performs worse in some cases.

RGGA Part seems to not work well because it seems that it is not able to optimize high-variance virtual machines with low-variance virtual machines together. The normal RGGA algorithm is able to optimize both types of virtual machines together. Thus, RGGA Part seems to not do as well.

As long as the high-variance virtual machines are packed on their own server, then the total number of servers over capacity should be fewer.

4.4 Conclusion

The effect of virtual machine variance on the Virtual Machine Assignment Problem is discussed. If a system administrator knows how much virtual machines vary, he or she can use this information to pack virtual machines more efficiently. RGGA has been shown to pack virtual machines with low variance very well, outperforming other heuristics used for comparison. Even when some virtual machines have low variance and some virtual machines have high variance, RGGA can pack the low-variance virtual machines naturally still outperforms other heuristics used for comparison. However, if all virtual machines have loads with high variance, then all packing algorithms considered in this section perform similarly.

4.5 Future Work

The results presented here in this paper have been shown empirically by running experiments. We did not develop any type of mathematical formula that would help to determine which packing algorithm to use for which problems. A more exhaustive search or proof should be done to find out in general which packing algorithms should be used when.

Moreover, even though RGGA works well for problems which have low variances, we have not discovered an algorithm which handles virtual machine assignment problems with high variance. A proof of existence or nonexistence of such an algorithm would further future research in this area.

Chapter 5

Virtual Machine Migration

In previous chapters, we defined the Virtual Machine Assignment Problem and showed how it can be used to place virtual machines onto servers. However, if virtual machines can be migrated, then it may not be so important to pack virtual machines tightly ahead of time. After discussing the relevance of virtual machine migration to the Virtual Machine Packing Problem we propose a load redistribution algorithm. We present results showing the number of servers yielded when running a virtual machine packing algorithm followed by a load redistribution algorithm. For some configurations, load redistribution algorithms cannot make up for a bad initial packing. However, for other configurations, load redistribution algorithms can quickly adjust to make up for a poor initial packing.

5.1 Introduction

This thesis has investigated the problem of packing virtual machines onto servers. This problem is important because of the large amount of electricity used in data centers. A good packing of virtual machines onto servers will decrease the number of servers in utilization, reducing energy and hardware costs.

The algorithms presented thus far might, for example, be applied at the beginning of a day at an e-commerce web site. At the beginning of the day, there are low loads on servers at the web site, and system administrators have more flexibility to move around virtual machines between servers. However, after the day has started, system administrators may not be able to move around virtual machine as freely because servers need to be dedicated to processing customer transactions, and not performing migrations between virtual machines.

While exploring VMAP, we have largely ignored live migration, which is the transfer of a virtual machine from one host to another without downtime between the hosts. Using live migration, it is possible to redistribute load between servers outside of initial virtual machine packings.

One assumption we have made so far is that a good initial packing is important to virtual machine packings. However, it might be possible for system administrators to do some live migrations during the day, reducing load on servers. Because it might be possible to use live migration technology to reduce the number of servers after virtual machines are deployed, one might question the relevance of doing a good packing. If live migration is a tool which can be used to swap virtual machines between servers, some would ask why doing a good initial packing is even important.

The purpose of our experiments was to analyze the feasibility of using live migration technology to swap virtual machines after an initial packing has been made. We performed some experiments where extra servers existed where virtual machines could be swapped in the case of high loads. We analyzed how many servers were in use after different runs using different inverse cumulative distribution function values.

In our experiments, first, we generate an initial assignment of virtual machines to servers. After this initial assignment of virtual machines is generated, loads are assigned to virtual machines. Virtual machines can then be moved in order to achieve a target percent of servers over capacity for the entire problem. After a target percent of servers over capacity is achieved by swapping virtual machines between servers, metrics such as the total number of servers in utilization can be gathered.

This chapter proceeds as follows. Section 5.2 discusses the live migration algorithm used in our results. Section 5.3 discusses how we set up our experiments in this chapter. Section 5.4 discusses the results of our experiments and our findings about how live migration could fit in with the virtual machine packing problem.

5.2 Load Redistribution Algorithm

Load redistribution algorithms swap virtual machines between servers. They move virtual machines from more heavily loaded servers to less heavily loaded servers. We present a load redistribution algorithm, Live Migration Algorithm (LMA), that we use to redistribute load between servers.

The basic intuition behind LMA is to run RGGA first. Afterward, LMA swaps enough servers from servers which are over capacity to servers which have never been over capacity in order to satisfy the target percent of servers over capacity. We illustrate this algorithm in Function 1.

5.3 Experimental Setup

The purpose of our experiments is to determine in what situations initial packing algorithms like RGGA are made invalid by live migration, and in what situations live migration algorithms cannot make up for inefficient preliminary packings. When conducting our experiments, there were a variety of steps:

Function 1 Redistributing Load Onto Servers

```
overcapacity ← calc_over_capacity()
prop_over ← (overcapacity − target) / overcapacity
servers_not_over ← get_servers_not_over_capacity()
servers_over ← get_servers_over()
count ← 0
for each s_i ∈ servers_over do
  get_servers_that_fit(servers_not_over, s_i) ← servers_that_fit
  if len(servers_that_fit) = 0 then
    new server()← server_to
  else
    servers_that_fit.sample() ← server_to
  end if
  swap smallest item from s_i to server_to
  count ← count + 1
  if count < prop_over * size(servers_over) then
    break
  end if
end for
```

1. RGGA is run on a problem and an initial packing solution is developed.

2. Loads on virtual machines are sampled and assigned from their respective load distributions.

3. LMA is used to migrate virtual machines from servers which are over capacity.

4. Metrics such as the total number of servers used are calculated.

In Chapters 3 and 4, we measured success by number of servers found for a particular percent of servers over capacity. With the option of live migration, it is possible to achieve any particular percent of servers over capacity by swapping virtual machines from servers which are heavily loaded to servers which have fewer virtual machines. Therefore, when performing our experiments, we consider the target percent of servers over capacity a value that system administrators can set. The inverse cumulative distribution function (icdf) value that should be used minimizes the number of servers for a particular target percent of servers over capacity. In our results, we focus on finding the particular inverse cumulative

distribution value that should be used with RGGA for a particular target percent of servers over capacity.

We performed experiments for each icdf value between 0.02 and 0.98 increasing with an interval of 0.02. We used target values of percent of servers over capacity between 0 and 1 increasing with an interval of 0.02. We used the same data set as was used in Chapter 4 and performed experiments generally in the same manner, except LMA was run after a solution from RGGA was generated in order to redistribute virtual machines after loads were observed. We analyzed how many servers were found after LMA corrected packings so that the y had the correct number of servers.

We recognize that the assumption that migration comes at no cost is not realistic. While recent advances in VM migration technology [11] make it possible for a VM instance to change host nearly instentaneously, migration does consume network resources. It is not clear whether the pay-off of extra migrations would justify added cost. In our experiments, because we have no basis on how many live migrations should be done, we allow an unbounded number of live migrations to perform, and report the number of live migrations done.

We ran three different experiments in Section 5.4. The three different experiments are for virtual machines which have low, medium and high variance in their loads. We wish to explore which icdf values should be used for the three different types of problems.

5.4 Results

In our experiments, we wished to find for differing target percents of servers over capacity, which icdf value should be used. We start our discussion for low-variance problems. We show the the number of servers found for a low-variance problem in Figure 5.1.

As mentioned earlier, system administrators can decide on a target percent of servers over capacity. With the target percent of servers over capacity in mind, they can decide which icdf value to use that will yield them the fewest number of servers, which in tern,

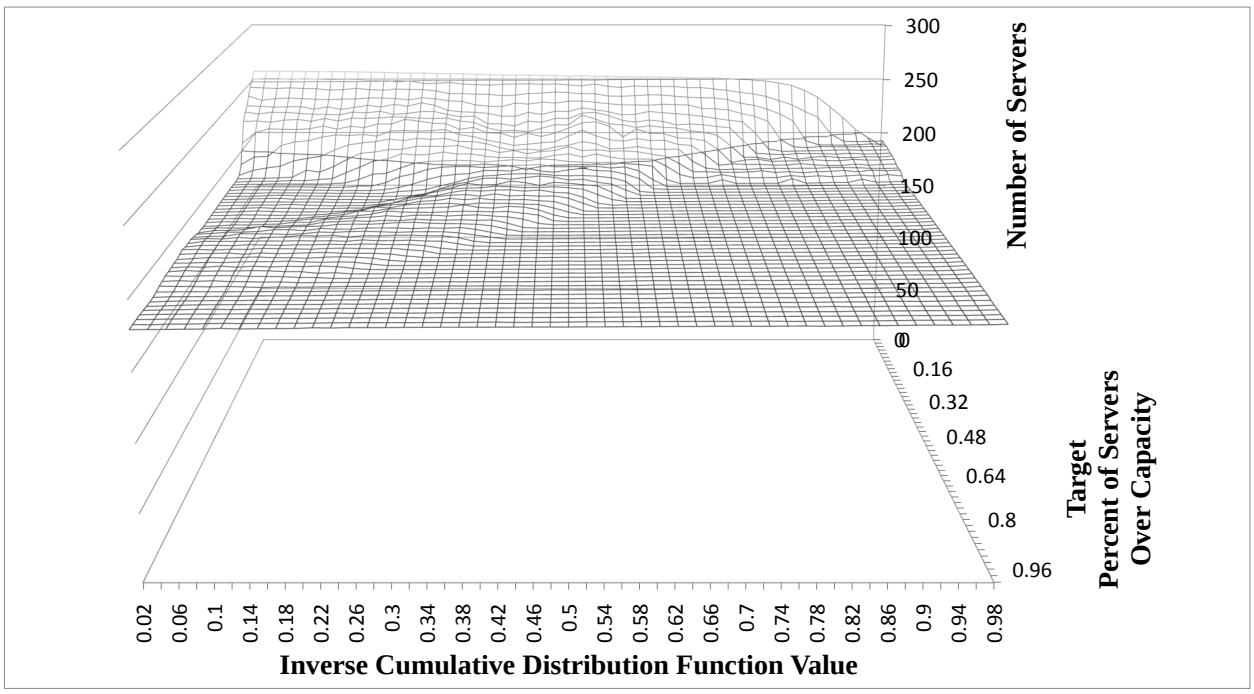

Figure 5.1: A surface graph showing the number of servers yielded by running RGGA at different icdf values and different targets with low variance.

saves the most money. Therefore, in our analysis, we will consider this scenario, which is finding the best icdf value value for given values for target percent of servers over capacity.

Figure 5.1 shows that for low values for target percent of servers over capacity, the best icdf value to use is 0.98. However, if the percent of servers over capacity can be higher, such as more than 0.1, then a low icdf value can be used.

This result means that live migration cannot compensate for a bad initial packing if a low percent of servers over capacity is desired. The best option is to use RGGA to develop an assignment of VMs to servers with a high icdf value, and do fewer swaps. This means that a good initial packing is needed for some configurations if virtual machines have low variance.

Next, we performed a similar experiment for virtual machines with medium variance. We wished to find, generally, for differing target percents of servers over capacity, which icdf value should be used. We show our results in Figure 5.2.

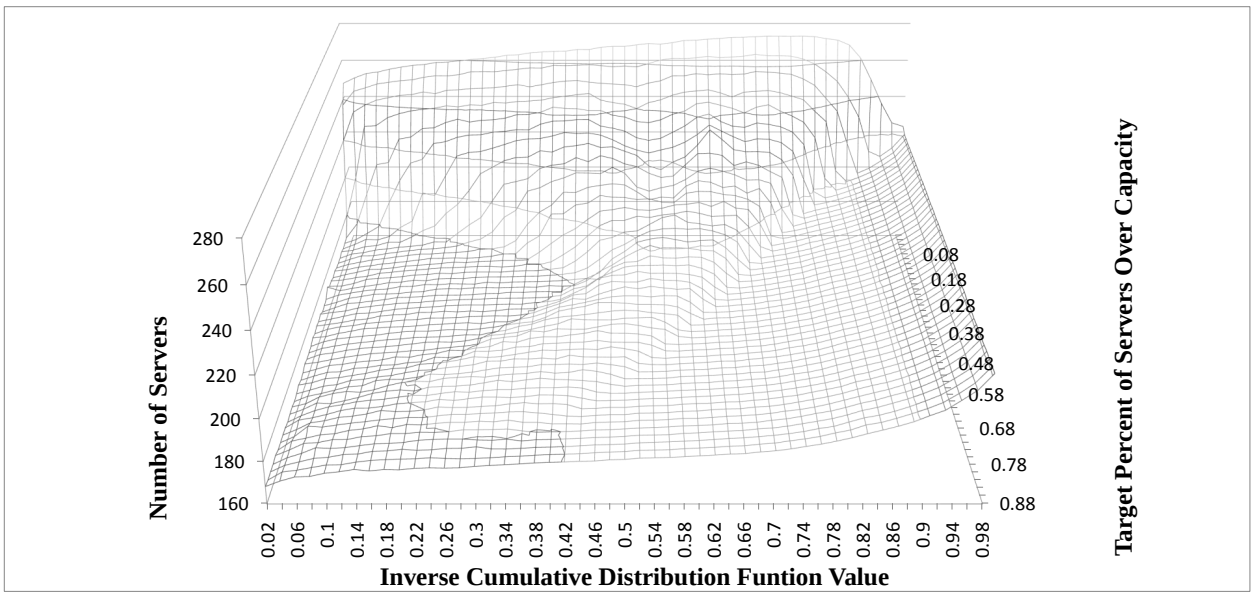

Figure 5.2: A surface graph showing the number of servers yielded by running RGGA at different inverse cumulative distribution values and different targets with average variance.

Figure 5.2 shows, again, that for a low target percent of servers over capacity, the best option is to use a low icdf value value. However, Figure 5.2 suggests that for moderately higher values for percent of servers over capacity, it is actually better to use a low icdf value value and swap. Nonetheless, there are circumstances in which system administrators would desire to have very low values for percent of servers over capacity. In these situations, it seems that the best option is to use a virtual machine packing algorithm with a , instead of merely swapping servers with live migration.

In our third and last experiment, we wondered what would happen if loads on virtual machines had high variances. We increased the variance on virtual machines from what was shown in Figure 5.2 by a factor of four, and ran the same experiment as was shown previously, except for higher variance values. We show the results of this experiment in Figure 5.3.

Again, in this graph, we see that for very high values for percent of servers over capacity, the best alternative is to use a low icdf value. However, again, if the value of target percent of servers over capacity is higher, then the best alternative may be to consolidate virtual machines and merely use live migration.

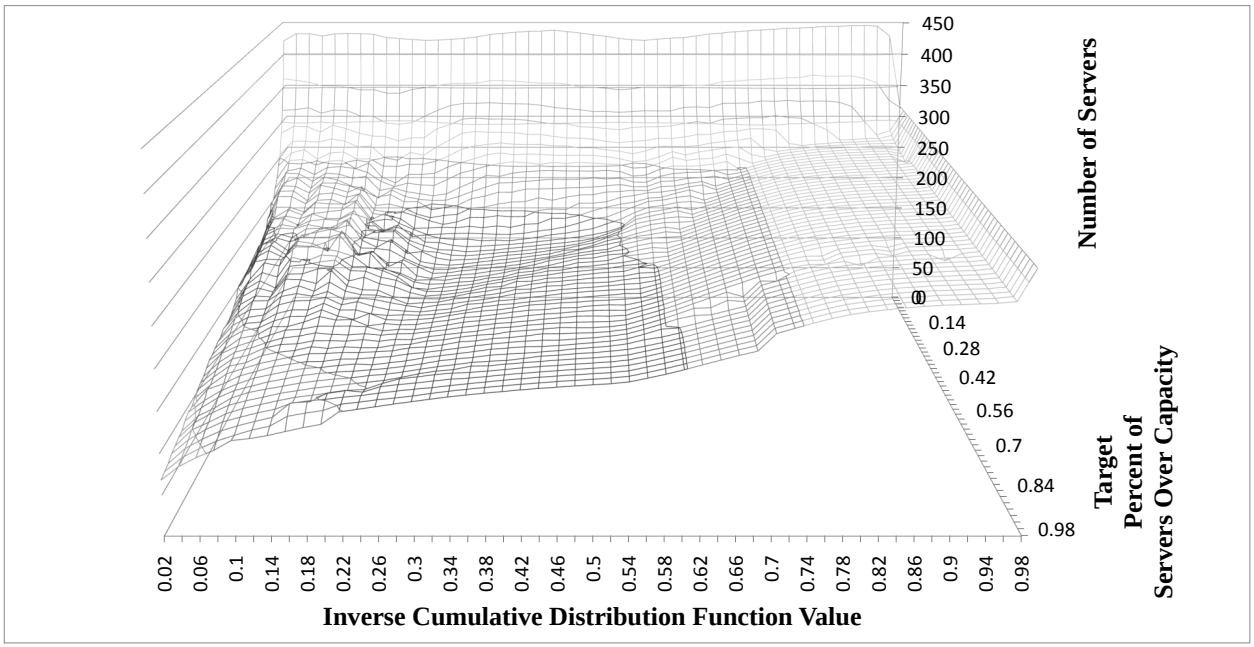

Figure 5.3: A surface graph showing the number of servers yielded by running RGGA at different icdf values and different targets with high variance.

The general trend from low variance to high variance seems to show that when virtual machines with high variance are packed, the packing algorithm can exploit the variance more with swaps. However, when variance is low, and there is less uncertainty in the loads of virtual machine, then less can be exploited by doing an initial tight packing and moving virtual machines which send the packing over to other virtual machines.

For reference, we show the number of migrations needed to achieve the number of servers shown. We show the number of swaps needed in Figures 5.4, 5.5 and 5.6. These figures all look similar, and what we expected. There are fewer swaps in general for low-variance problems and more swaps in general for high-variance problems.

Some conclusions are:

1. If the target percent of servers over capacity is low (near zero), then a very high icdf value should be used.

2. If the number of live migrations done during run-time is a factor during deployment, then a high icdf value should be used.

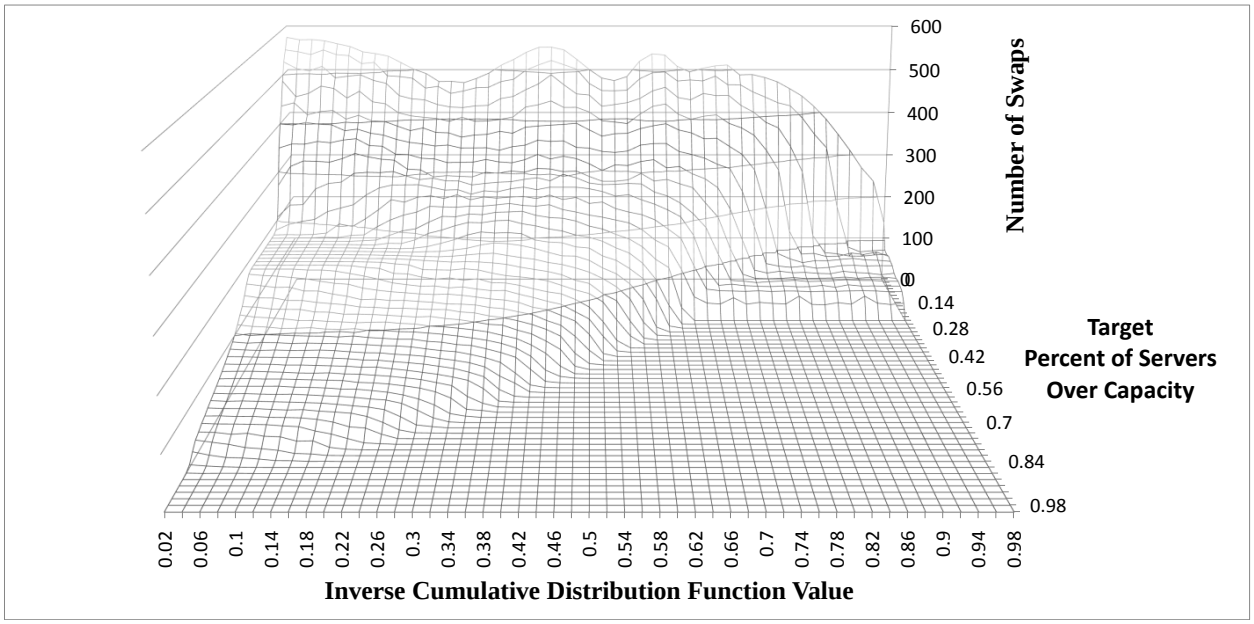

Figure 5.4: A surface graph showing the number of servers yielded by running RGGA at different icdf values and different values of target percent of servers over capacity with low variance.

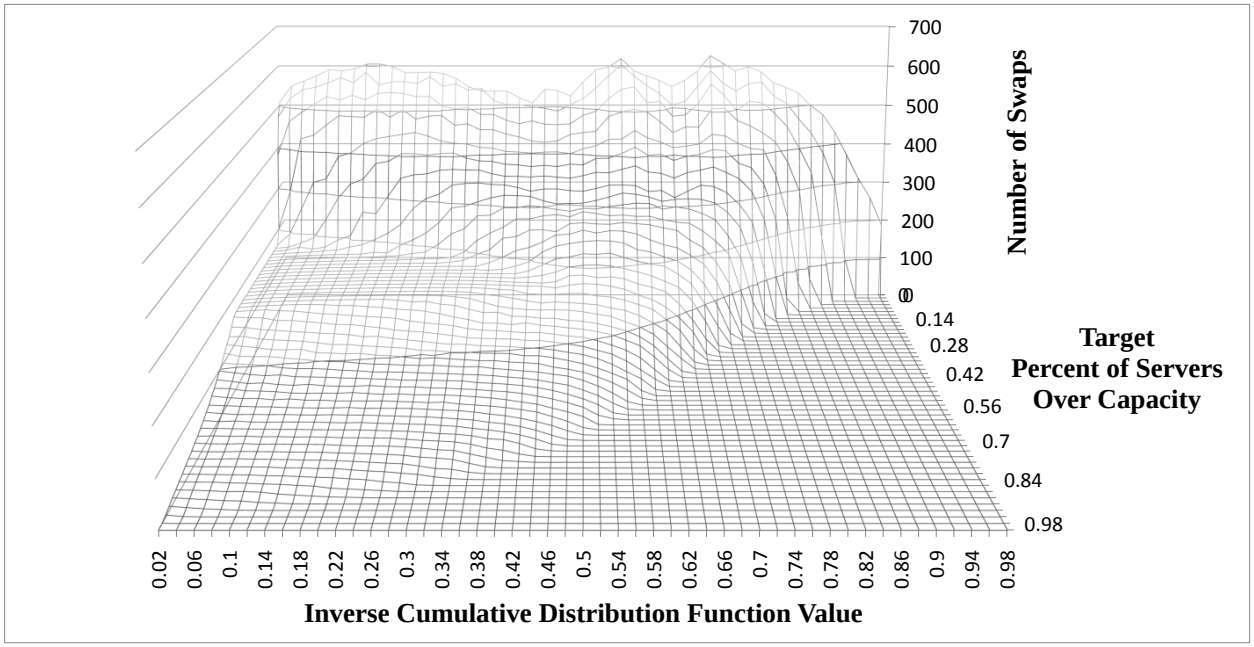

Figure 5.5: A surface graph showing the number of servers yielded by running RGGA at different icdf values and different values of target percent of servers over capacity with medium variance.

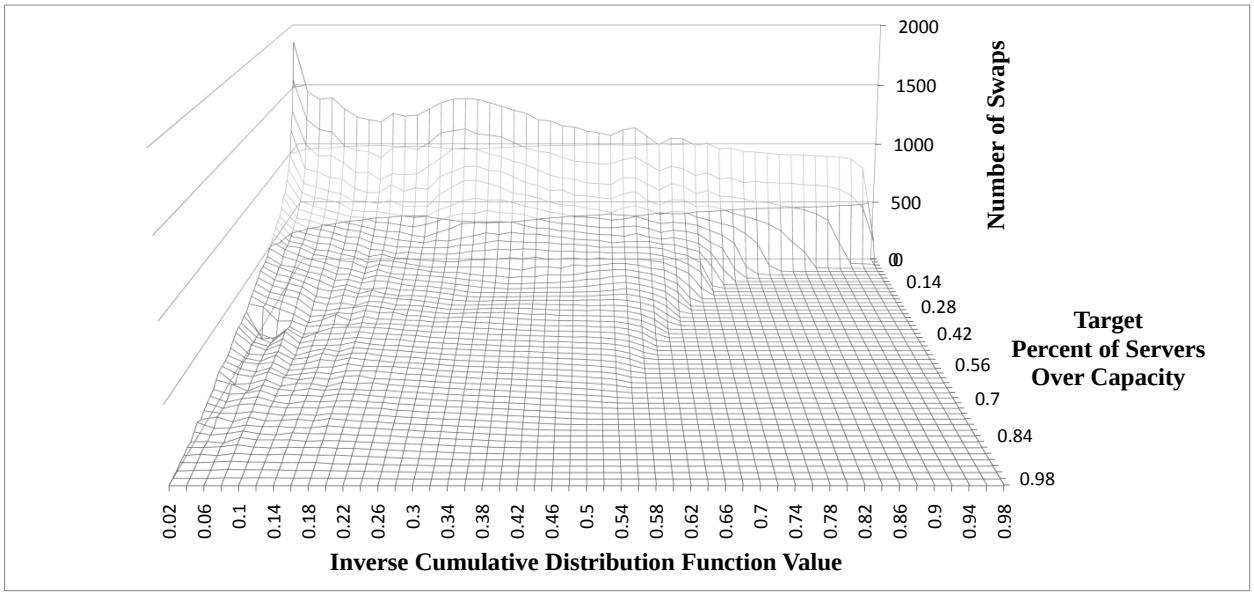

Figure 5.6: A surface graph showing the number of servers yielded by running RGGA at different icdf values and different values of target percent of servers over capacity with high variance.

3. If the number of swaps done during the run-time of virtual machines does not matter, and system administrators can tolerate some servers over capacity, then a packing with a low icdf value with load redistribution could be used.

5.5 Conclusion

At times, it may be possible to swap virtual machines between servers. If swapping virtual machines between servers is viable, then the number of servers in use after swapping servers is still important to consider. We present a reassignment algorithm (LMA) that can be used to swap virtual machines between servers.

We present results that show the results of swapping virtual machines at different configuration settings. We show that when the target percent of servers over capacity is very small (around zero percent), then a high icdf value should be used. However, if the target percent of servers over capacity is higher, then a tight packing using a very low icdf value with swapping may perform better than a packing with a high icdf value.

We also show in this paper that if loads on virtual machines have high variance then a load redistribution algorithm can take advantage of observed loads by redistributing virtual machines with observed loads to other servers. This allows the load redistribution algorithm to essentially see the load and fix the packing, achieving any arbitrary amount of percent of servers over capacity.

Chapter 6

Conclusions and Future Work

6.1 Conclusion

This thesis has investigated the problem of packing virtual machines onto servers. This problem is important because of the large amount of electricity used in data centers. A good packing of virtual machines onto servers will reduce the number of servers in utilization, reducing energy and hardware costs.

We defined the Multi-Capacity Bin Packing Problem and showed its applicability to the Virtual Machine Packing Problem (VMAP). Afterward, we defined Reordering Grouping Genetic Algorithm (RGGA) and showed its applicability to the Virtual Machine Assignment Problem. We showed that Reordering Grouping Genetic Algorithm is fast, accurate, and applicable to the Virtual Machine Packing Problem.

We proposed a model for virtual machine packing. Specifically, we added the idea that loads on virtual machines are not deterministic as in the Bin Packing Problem, but rather probabilistic, and can be represented by probabilistic distributions. We gave results for preliminary experiments for this model.

Third, we investigated the effect of what happens when system administrators know something about the variance of loads on virtual machines. We found that Reordering Grouping Genetic Algorithm performs well when variance of loads on virtual machines is low.

Lastly, we investigated the relevance of our research in the face of the availability of extra servers and live migration. We found that it is still important to do a good initial

packing even if it is possible to migrate some virtual machines to other servers. It seems that even if virtual machines are swapped from servers, this swapping is not as efficient as what could be found by an optimization algorithm.

Some of the ideas presented in this paper led us to experiment with different variants of RGGA.

6.1.1 Variants of RGGA

Our first observation was that RGGA generally packed some servers very tightly and other servers very loosely. We wanted to spread the free space around so that all servers are packed to about the same point.

The first variant (RGGA w/ Repack) takes the solution that RGGA returns and rebalances the solution to balance loads between servers. It first determines the solution that RGGA would have given, and then runs a second genetic algorithm. This second genetic algorithm starts with the same population as was returned by the previous algorithm. The goal of this algorithm is to spread virtual machines out as evenly as possible between servers. This is done by modifying Equation 2.3. By setting $\kappa = \frac{1}{2}$, solutions are rewarded most when servers are most spread out. We only keep solutions that had the same number of servers as were found in the original solution.

The motivation for this approach is to create a solution that has the exact same number of servers as the solution found by RGGA, but spread out the virtual machines so that there is an equal amount of free space in each server. This way, there is a bit of room for virtual machines to "spill over" to if they end up using more resources than is initially allotted to them.

The second approach (RGGA w/ Distro) is motivated by the idea that the packing algorithm may be able to do better with a knowledge of the load distribution instead of merely expectations on the loads. What we do in the third algorithm is add the respective

distributions together using known methods for adding probabilistic distributions, and then sample an expectation from the new, convolved distribution.

Let $\boldsymbol{N}$ be all resource distributions corresponding to one resource of one virtual machine. Let $\boldsymbol{\mu}$ be the mean for all resource distributions corresponding to $\boldsymbol{N}$ and $\boldsymbol{\sigma^2}$ be the variance for all resource distributions corresponding $\boldsymbol{N}$. Let $icdf(N, p)$ be the icdf function for distribution N and value p. Normally, when applying a packing algorithm, we applied the following algorithm for a vector $\boldsymbol{N}$ of resource distributions corresponding to a VM:

$$est = \sum_{N \in \boldsymbol{N}} icdf(N, p) \tag{6.1}$$

However, because in our experiments, loads could be modeled well with the normal distribution, instead of applying the above equation, we add together the seperate load distributions before taking the ICDF estimate.

$$est = icdf(\sum_{i=1}^{|\boldsymbol{N}|} \boldsymbol{N_i}, p) \tag{6.2}$$

$$est = icdf(Normal(\sum_{i=1}^{|\boldsymbol{\mu}|} \mu_i, \sum_{i=1}^{|\boldsymbol{\sigma^2}|} \sigma_i^2), p) \tag{6.3}$$

In our tests, we used normal distributions, which add together easily. Assume that one particular load on computer c_1 is distributed as $N_1(\mu_1, \sigma_1^2)$. Assume also that the one particular load on computer c_2 is distributed as $N_2(\mu_2, \sigma_2^2)$. The added distribution of $N_1 + N_2$ is $N_3(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$. In the third approach, we use an expectation of the added distribution using the methods proposed earlier. This expectation will be used for all packing purposes. The difference between this approach and the earlier approaches is that in the earlier approaches, the expectation is taken first, and the expectations are added together. In this approach, the loads are added together first, and then the expectation is taken.

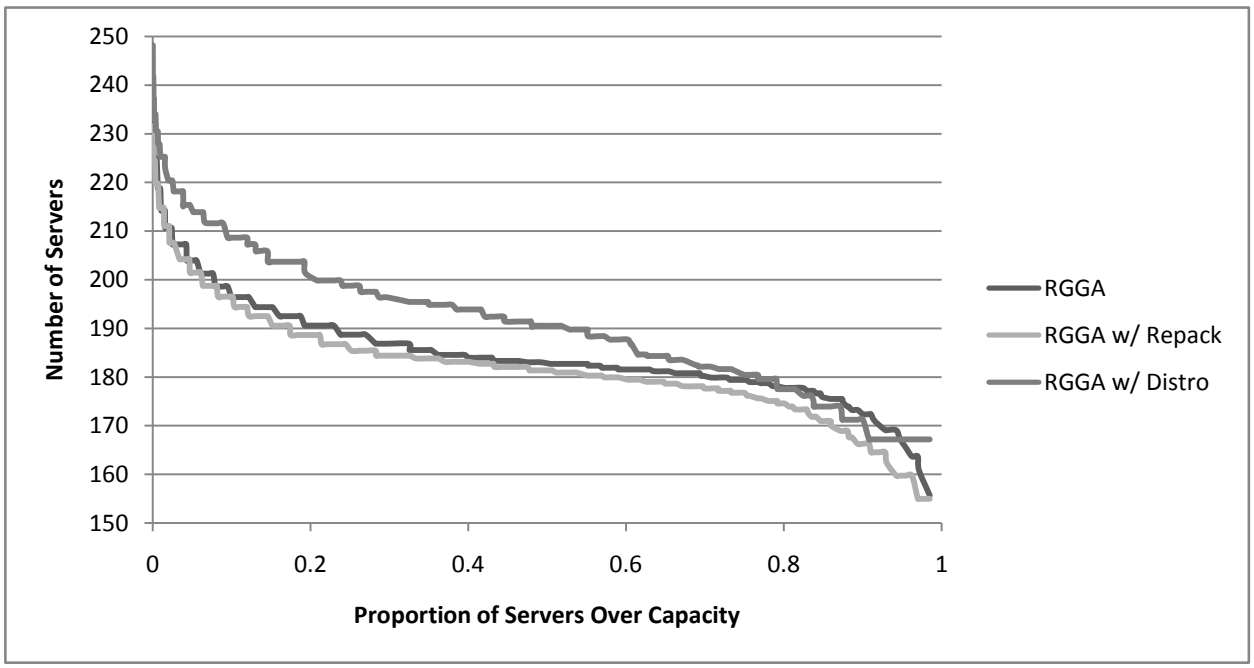

Figure 6.1: The number of servers found for varying algorithms vs percent of servers over capacity. Half of VMs have high variance. Half of VMs have low variance.

We show in Figure 6.1 the performance of these new variants compared with the other algorithms we used. Figure 6.1 shows that the new variants on the algorithms proposed did not outperform RGGA. The only variant on RGGA that shows promise is RGGA with Repack. But, even that solution is only marginally better than RGGA.

In conclusion, a good initial packing is important to consolidate energy in packing virtual machines. How virtual machines are packed onto servers makes a difference on the number of servers used. More balanced packings tend to use fewer servers.

6.2 Future Work

The work in this paper lays the foundation for other organizations to pack virtual machines onto servers. Even though we did preliminary tests by packing real virtual machines onto servers, these loads were largely contrived and made up. We have no basis to justify that the loads we used for virtual machine assignment are similar to loads found on virtual machines

in real deployments. Even though the results I present in this thesis show promise, we have not tested the concepts presented on a real industrial deployment.

Ideally, future work should be done by an organization that has data describing loads on virtual machines in a real deployment. Using this data describing loads on real virtual machines, the research should focus on how the concepts presented in this paper fair in modeling loads on real virtual machines, what modifications needed to be done, and other things that were generally unforeseeable without real data from a real deployment and the ability to test algorithms with a real deployment.

Moreover, loads on virtual machines in real deployments invariably change throughout time. Our algorithm here assumes a static distribution on virtual machines throughout time. In the model proposed here, the loads on virtual machines can be described by a probabilistic distribution, but this probabilistic distribution stays the same through time. We have proposed no way of modeling a changing probabilistic distribution as a day progresses. Future research could focus in this area, discovering what happens when the probabilistic distribution that describes virtual machine load changes throughout time.

Load on servers have historically been described using queueing theory. Future research can focus on how to use queueing theory to develop probabilistic distributions for the loads on virtual machines that can be used with the model presented in this thesis. This type of research would make the research presented here more applicable and usable to industry.

References

[1] S. Agarwala, F. Alegre, K. Schwan, and J. Mehalingham. E2EProf: Automated End-to-End Performance Management for Enterprise Systems. In *DSN '07: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 749–758. IEEE Computer Society, Washington, DC, USA, 2007.

[2] G. Aggarwal, R. Motwani, and A. Zhu. The Load Rebalancing Problem. *ACM Symposium on Parallel Algorithms and Architectures*, 15, 2003.

[3] A. C. F. Alvim, C. C. Ribeiro, F. Glover, and D. J. Aloise. A Hybrid Improvment Heuristic For The One-Dimensional Bin Packing Problem. *Journal of Heuristics*, 10:4–27, 2004.

[4] M. Bailey, T. Rostrom, and J. Ekstrom. Operating System Power Dependencies. In *Int. Computer Measurement Group*, pp. 15–20. 2008.

[5] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pp. 337–350. USENIX Association, Berkeley, CA, USA, 2008.

[6] IT Knowledge Exchange. Virtual Machines Per Server: A Viable Metric For Hardware Selection? 2008 (Accessed October 28, 2009). `http://itknowledgeexchange.techtarget.com/server-farm/virtual-machines`.

[7] E. Faulkenaeur. A Hybrid Grouping Genetic Algorithm for Bin Packing. *Journal of Heuristics*, 2:5–30, 1996.

[8] K. Fleszar and K. S. Hindi. New Heuristics For One-Dimensional Bin Packing. *Computers and Operations Research*, 29:821–839, 2002.

[9] T. C. Wilcox Jr. Dynamic Load Balancing of Virtual Machines Hosted on Xen. 2009.

[10] N. Karmarkar and R. M. Karp. An Efficient Approximation Scheme For The One-Dimensional Bin-Packing Problem. *23rd Annual Symposium on Foundations of Computer Science.*, pp. 312–320, 1982.

[11] C. C. Keir, S. Hanson, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *In Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI*, pp. 273–286. 2005.

[12] J. G. Koomey. Estimating Total Power Consumption By Servers In The U.S. and The World. *Presented at The Environmental Protection Agency Stakeholder Workshop at Santa Clara Convention Center*, 2007.

[13] W. Leinberger, G. Karypis, and V. Kumar. Multi-Capacity Bin Packing Algorithms with Applications to Job Scheduling under Multiple Constraints. In *ICPP '99: Proceedings of the 1999 International Conference on Parallel Processing*, p. 404. IEEE Computer Society, Washington, DC, USA, 1999.

[14] H. Lima and T. Yakawa. A New Design of Genetic Algorithm For Bin Packing. *Evolutionary Computation, 2003. The 2003 Congress on Evolutionary Computation*, 2:1044–1049, 2003.

[15] M. Mitchell. *An Introduction To Genetic Algorithms*. MIT Press, 1998.

[16] R. Nathuji and K. Schwan. VirtualPower: Coordinated Power Management in Virtualized Enterprise Systems. In *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating systems principles*, pp. 265–278. ACM, New York, NY, USA, 2007.

[17] N. J. Radcliffe. Forma Analysis and Random Respectful Recombination. *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1991.

[18] P. Rohlfshagen and J. Bullinaria. A Genetic Algorithm with Exon Shuffling Crossover for Hard Bin Packing Problems. *Proceedings of Genetic And Evolutionary Computation Conference*, 9:1365–1371, 2007.

[19] A. Scholl, R. Klein, and C. J. Bison. A Fast Hybrid Procedure for Exactly Solving the One-Dimensional Bin Packing Problem. *Computers and Operations Research*, 24(7):627–645, 1997.

[20] L. P. Slothouber. A Model of Web Server Performance. In *In Proceedings of the Fifth International World Wide Web Conference*. 1996.

[21] Y. Song, H. Wang, Y. Li, B. Feng, and Y. Sun. Multi-Tiered On-Demand Resource Scheduling for VM-Based Data Center. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 148–155. IEEE Computer Society, Washington, DC, USA, 2009.

[22] Y. Song, Y. Zhang, and Y. Sun. Utility Analysis for Internet-Oriented Server Consolidation in VM-Based Data Centers. *IEEE International Conference on Cluster Computing*, 0:xvii, 2009.

[23] S. Srikantaiah, A. Kansal, and F. Zhao. Energy Aware Consolidation for Cloud Computing. In *Proceedings of HotPower '08 Workshop on Power Aware Computing and Systems*. USENIX, 2008.

[24] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova. Resource Allocation Using Virtual Clusters. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 260–267. IEEE Computer Society, Washington, DC, USA, 2009.

[25] C. W. Szeto. On the Modeling of WWW Request Arrivals. In *Proceedings of the 1999 International Workshops on Parallel Processing*, p. 248. IEEE Computer Society, Washington, DC, USA, 1999.

[26] VMware. VMware Green IT Energy Efficiency, Reduce Energy Costs with Virtualization. 2009 (Accessed October 18, 2009). `http://www.vmware.com/virtualization/green-it/`.

[27] VMWare. Server Consolidation Overview, Building a Virtual Infrastructure. 2009 (Accessed October 5, 2009). `http://www.vmware.com/solutions/consolidation/consolidate.html`.

[28] D. Wilcox, A. McNabb, K. Seppi, and K. Flanagan. A Model for Initial Virtual Machine Assignment. *To be Published in Proceedings of The First International Conference on Cloud Computing, GRIDs, and Virtualization*, 2010.

[29] D. Wilcox, A. McNabb, K. Seppi, and K. Flanagan. Virtual Machine Assignment with a Reordering Grouping Genetic Algorithm. *To be Published in Proceedings of Genetic And Evolutionary Computation Conference*, 2010.