2019 IEEE International Conference on Autonomic Computing (ICAC)

Autonomic cloud placement of mixed workload: An adaptive bin packing algorithm

Asser N. Tantawi and Malgorzata Steinder
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598
Email: tantawi, steinder@us.ibm.com

***Abstract*—Cloud computing offers a platform where virtual entities, such as virtual machines, containers, and pods, are hosted in a physical infrastructure. Such virtual entities request resources, such as CPU, memory, and GPU, among other constraints. The cloud placement engine, also referred to as the scheduler, needs to place, in real time, such virtual entities in the cloud. Typically, resource demand is heterogeneous and the mix varies over time. Therefore, the scheduler needs to change its placement policy dynamically in order to accommodate the change in the mixed demand, resulting in lower rejection probability. A novel, autonomic, Adaptive Bin Packing (ABP) algorithm which attempts to equalize measures of variability in the demand and the allocated resources in the cloud, without the need to set any configuration, is introduced. ABP is compared to simplistic, extreme packing policies (spread and pack) as well an optimized packing policy. Experimental results based on simulations are presented, and the behavior of ABP and its adaptability to the demand mix is demonstrated. Further, ABP performs close to the optimized policy, yet evolves to an extreme policy as the mix becomes homogeneous.**

***Keywords*-bin packing; cloud optimization; cloud placement; Kubernetes scheduler;**

I. Introduction

The cloud placement problem (CPP) is about assigning Virtual Entities (VE), such as Virtual Machines (VM), Kubernetes pods [1], and data volumes, to Physical Entities (PE), such as Physical Machines (PM), nodes, and storage devices, in the cloud infrastructure. A PE comprises a set of resources, such as CPU cores, memory, and disk storage, with some capacity. A request for a VE involves a specified demand for such resources. A stream of single or multiple (group of) VEs constitute the input to a placement engine, also known as scheduler, which assigns a VE to a PE, satisfying resource demand, as well as potentially other constraints, with a certain set of objectives. Placement constraints may involve PE preferences and/or affinity, and may be hard or soft. Placement objectives involve system-related objectives, such as load balancing, and user-related objectives, such as quality of placement in terms of satisfying soft constraints. CPP has been around for a while and has been the subject of a large amount of research [2].

The original CPP involved only the placement of a single VE. This gives rise to an (online) bin packing problem with multiple dimensions, one for each resource type. Then, multiple VEs were considered along with their communication needs, resulting in the so-called traffic-aware placement problem which is naturally quadratic. For more general CPP, involving additional constraints, such as location, availability, and security constraints, we find a variety of techniques, ranging from heuristic-based, simulation, to evolutionary algorithms, for solving such a large scale combinatorial optimization problem. A biased sampling algorithm is presented [3]. Others approaches include a topology-aware mapping algorithm [4].

As for the area of bin packing, there has been a large amount of research, but we are here concerned with applications of bin packing in distributed systems and cloud computing, in particular. A survey on multidimensional bin packing algorithms is provided [5]. Resource allocation using bin packing techniques have been employed to investigate overcommitment in cloud services [6].

In this paper, we focus on the multidimensional bin packing version of CPP. In particular, a VE specifies a set of, potentially dependent, resource demand values. Examples are offerings of a set of standard VE sizes and/or user-specified custom sizes. The scheduler places one or multiple VEs at a time in such a way to minimize the probability of failing to find a placement. Additional objectives, such as load balancing, may have to be taken into consideration when making a placement decision. The distribution of resource demand varies over time, hence the scheduler needs to adapt its placement policy, accordingly. This may be the result of having several types of workloads and/or unpredictable changes over time. Simplistic, extreme, and standard policies, such as pack and spread may lead to resource fragmentation and rejection of large-sized and/or disproportional VEs. Moreover, a static choice of a placement strategy for a given workload mix may prove inefficient as the mix changes. The challenge is to keep the time to make a placement decision relatively short, in order to accommodate a high throughput. Hence, a simple, approximate algorithm for this problem of a mixed and varied demand, multidimensional bin packing problem is sought.

We provide a novel algorithm for adaptive bin packing (ABP). In order to adapt to the mixed workload, we introduce a measure of variability for both VE demand and PE usage in the cluster. The measures are based on multivariate coefficient of variation [7], which we conjecture are simple to calculate and capture the variability, without having to work with probability distributions. ABP is built

2474-0756/19/$31.00 ©2019 IEEE
DOI 10.1109/ICAC.2019.00030

on a simple heuristics, namely, when making a placement decision, attempt to equalize the variability in the demand to the variability in the allocated resources across the physical cluster. The implementation of ABP is rather straightforward. Using a cloud simulation environment, we compare ABP to existing simple bin packing policies as well as an optimized policy and demonstrate its merits.

II. Problem Statement

Consider a physical infrastructure comprising a collection of PEs represented by the set $\mathcal{N} = \{p_1, p_2, \cdots, p_N\}$. We also refer to it as the cloud, the data center, or the cluster, while referring to a particular element as a node. A node comprises R-type resources represented by the set $\mathcal{R} = \{t_1, t_2, \cdots, t_R\}$. We denote resource on node p_n of type t_r as $s_{n,r}$, where $n = 1, 2, \cdots, N$ and $r = 1, 2, \cdots, R$, having capacity $c_{n,r}$.

Consider an arrival of an LE request with resource demand d_r for resource type $t_r \in \mathcal{R}$. At the time of arrival, let resource $s_{n,r}$ be allocated (also referred to as used) an amount denoted by $g_{n,r}$, and equivalently an available resource amount of $a_{n,r} = c_{n,r} - g_{n,r}$, where $a_{n,r} \geq 0$.

We use the notation that boldfaced variables represent vectors or matrices, depending on the dimensionality of the variable. Hence, we have matrices $\mathbf{c}$, $\mathbf{g}$, and $\mathbf{a}$, for resource capacity, allocation, and available, respectively, and vector $\mathbf{d}$, for LE resource demand. A matrix with a subscript refers to the row with the subscripted index in the matrix. For example, $\mathbf{g}_n$ is the vector of allocation for the resources on node n.

Let $\mathbf{g}^+(n;\mathbf{d})$ and $\mathbf{a}^+(n;\mathbf{d})$ denote resource allocation and available after placing demand $\mathbf{d}$ on node n, respectively. Thus, we have

$$\mathbf{g}_i^+(n;\mathbf{d}) = \begin{cases} \mathbf{g}_i, & i \neq n, \\ \mathbf{g}_i + \mathbf{d}, & i = n, \end{cases}$$

$$\mathbf{a}_i^+(n;\mathbf{d}) = \begin{cases} \mathbf{a}_i, & i \neq n, \\ \mathbf{a}_i - \mathbf{d}, & i = n. \end{cases}$$

Denote the utilization of resource $s_{i,r}$, after allocating demand $\mathbf{d}$ on node n, by $u_{i,r}(n;\mathbf{d})$, where $i, n = 1, 2, \cdots, N$ and $r = 1, 2, \cdots, R$. It is given by the vector

$$\mathbf{u}(n;\mathbf{d}) = \mathbf{g}^+(n;\mathbf{d})/\mathbf{c},$$

where the division operation is applied element-wise, $\mathbf{0} \leq \mathbf{u}(n;\mathbf{d}) \leq \mathbf{1}$. Denote the average and standard deviation of the utilization $\mathbf{u}(n;\mathbf{d})$ across the cluster $\mathcal{N}$ (i.e., taken column-wise) by the vectors $\boldsymbol{\mu}(n;\mathbf{d})$ and $\boldsymbol{\sigma}(n;\mathbf{d})$, respectively. For convenience, we define the coefficient of variation vector $\boldsymbol{\gamma}(n;\mathbf{d}) = \boldsymbol{\sigma}(n;\mathbf{d})/\boldsymbol{\mu}(n;\mathbf{d})$, where the division is taken element-wise and $\gamma_r = 0$ if $\mu_r = 0$, $r = 1, 2, \cdots, R$.

Define the constraint function, capturing the feasibility of allocating an arriving LE with demand $\mathbf{d}$ on node n, as

$$\psi(n;\mathbf{d},\mathbf{a}) = \mathbf{a}^+(n;\mathbf{d}) \ \geq \mathbf{0}. \tag{1}$$

Let $\phi(n;\mathbf{d},\mathbf{a})$ be the objective function, evaluated for placing an arriving LE with demand $\mathbf{d}$ on node n given state $\mathbf{a}$. An example of an objective function which balances load is denoted by $\phi_{lb}(\bullet)$. It may target a particular resource r, combine all resources, or combine a normalized measure, as $\phi_{lb}(n;\mathbf{d},\mathbf{a}) = \sigma_r(n;\mathbf{d})$, $\|\boldsymbol{\sigma}(n;\mathbf{d})\|$, or $\|\boldsymbol{\gamma}(n;\mathbf{d})\|$, respectively.

A. Placement Problem

The goal is to find an assignment of a PE, say p_n, to an arriving LE with demand $\mathbf{d}$, given a cluster state, $\mathbf{a}$, by solving the optimization problem,

$$\min_n \ \phi(n;\mathbf{d},\mathbf{a}), \quad s.t.\ \psi(n;\mathbf{d},\mathbf{a}) \geq \mathbf{0}, \tag{2}$$

$n = 1, 2, \cdots, N$.

If there is no feasible solution then the LE is rejected (lost). A *good* placement algorithm attempts to minimize the rejection probability over a given time horizon, in conjunction with other objectives, such as load balancing across the cluster.

Next, we consider two simple, well-known placement policies: $Pack$ and $Spread$. Then, we describe the $XBalance$ policy, which solves optimization problem 2 for a special objective function. In Section III, we present our ABP algorithm.

B. $Pack$ Policy

According to the $Pack$ policy, an arriving LE is placed on the node with the resulting highest allocation. There are several variations of this policy in the case of $R > 1$. Let us consider two variations: $Pack_r$ and $Pack_{dist}$. In $Pack_r$, resource type t_r is designated as the prime resource. Consequently, the selected p_n will have the minimum value of $(a_{n,r} - d_r) \geq 0$, irrespective of the other resource types, while satisfying condition 1. Whereas, in $Pack_{dist}$, a Cartesian distance is calculated. Thus, in the case of a homogeneous cluster, the selected p_n will have the maximum value of $\|\mathbf{u}(n;\mathbf{d})\|$, while satisfying condition 1. Note that dividing by the capacity normalizes quantities over the resource types. In general, the $Pack$ policy results in a skewed loading of the nodes in the cluster. It accommodates a mix of LE demands by making sure that some nodes possess large amounts of available resources for potentially arriving LEs with large demand. However, this skewed loading works against a desired load balanced cluster.

C. $Spread$ Policy

A $Spread$ policy behaves in a way opposite to the $Pack$ policy. Thus, a $Spread_r$ policy selects the p_n with the maximum value of $(a_{n,r} - d_r) \geq 0$. Similarly, a $Spread_{dist}$ selects the p_n with the the minimum value of $\|\mathbf{u}(n;\mathbf{d})\|$, in the case of a homogeneous cluster. In both cases, condition 1 is satisfied. In general, the $Spread$ policy balances the load across the cluster. However, it runs the risk of not being

able to accommodate a potentially arriving LE with large demand.

The $Pack$ and $Spread$ policies act as extremes. There are many other bin packing policies in the literature. We have particularly chosen these two policies for comparison because they represent the extreme cases and they are standard options in existing schedulers [1].

D. $XBalance$ Policy

We devised a generalized policy which results from solving optimization problem 2, using an objective function which is a linear combination of pack and spread across the various resources. As a result, one can selectively pack and/or spread load on resources in the cluster. We call this policy $XBalance$. In particular, define a weight vector $\mathbf{w}$, where the value of w_r is the weight of resource type t_r, and its sign the direction. In other words, $(w_r > 0)$ signifies balancing (i.e. spreading), $(w_r < 0)$ un-balancing (i.e. packing), and $(w_r = 0)$ irrelevant, respectively. Thus, an objective function of the $XBalance$ policy may be written as,

$$\phi(n; \mathbf{d}, \mathbf{a}) = \mathbf{w}^T \, \boldsymbol{\sigma}(n; \mathbf{d}).$$

Given the combinatorial complexity of solving problem 2, we employ the Biasing Sampling Algorithm (BSA) [3].

III. Solution

Consider optimization problem 2 where the objective function $\phi(n; \mathbf{d}, \mathbf{a})$ is a combined measure of load balancing and the probability of loss over a given time horizon. We conjecture that ABP is a heuristic algorithm which attempts to solve such a problem. In a nutshell, from the stream of arriving LE requests, one may learn the variability in demand for the various resources. Assuming that variability is time-homogeneous, one can predict demand variability of future requests based on past, observed variability. A particular resource type with high variability requires special care when placing a request in the cluster. In such a case, the placement policy should make room for future requests by packing requests on that resource type. As a consequence, the cluster would have nodes with high availability of the resource type in question, enough to accommodate future requests with high demand. On the other hand, observed low variability for a given resource type results in more spreading of the requests on that resource type, hence improving load balancing.

We define two measures of variability, one for the observed demand, and the other for the cluster loading. The ABP heuristic attempts to equate those two measures, when placing requests. Thus, ABP observes variability in workload demand, keeps track of variability in cluster resource usage, evaluates multi-dimensional (multi-resource) variability measures, and seeks placement resulting in equalized variability measure of workload and cluster. ABP uses biased, importance sampling for learning good placement decisions, without solving a massively large combinatorial problem.

A. Definitions

Consider the time, denoted by t, at which an LE request with demand $\mathbf{d}$ arrives to the system. Firstly, we describe the measure of variability in demand. Consider an arbitrary time period, up to and including t, over which demand statistics of arriving LEs are calculated. In practice, this period may be a sliding window or of infinite length where recent values are added with a smoothing factor. Instead of using absolute demand $\mathbf{d}$, we introduce relative demand $v_r = d_r/\overline{c}_r$, where $\overline{c}_r > 0$ is the maximum capacity of resource type t_r in the cluster. Let the average and covariance measures of relative demand over such a period be ν_r and $\upsilon_{r1,r2}$, respectively, for resource types $r, r1, r2 \in \{1, 2, \cdots, R\}$. Consequently, denote the relative demand average and covariance at time t by $\boldsymbol{\mu}^{dem}(t) = [\nu_r]$ and $\boldsymbol{\Sigma}^{dem}(t) = [[\upsilon_{r1,r2}]]$, respectively. Note that $\boldsymbol{\Sigma}^{dem}(t)$ is symmetric with the diagonal being the variance.

Secondly, we describe the measure of variability in the system, after the placement of the LE request on node n. Let $\mathbf{u}$(n) denote the resource utilization matrix. Denote the (marginal) average and covariance of $\mathbf{u}$(n) across the cluster $\mathcal{N}$ by $\boldsymbol{\mu}^{sys}(n)$ and $\boldsymbol{\Sigma}^{sys}(n)$, respectively. Again, $\boldsymbol{\Sigma}^{sys}(n)$ is symmetric with the diagonal being the variance.

In the special case of a single resource, i.e. $R = 1$, the (univariate) coefficient of variation, defined as the ratio of standard deviation over mean, may be used as the measure of variability. However, in the general case, we need to define a scalar (multivariate) coefficient of variation. There are several such definitions in the literature [7] that are beyond the scope of this paper. We selected a particular measure, known as the Albert and Zhang multivariate coefficient of variation for the reasons of (1) simplicity in computation complexity (no need to do matrix inversion), (2) capturing the correlation structure among the resources, and (3) preserving the scaling among the resource components based on the average usage. This measure is defined as

$$\boldsymbol{\gamma} = \sqrt{\frac{\boldsymbol{\mu}^T \, \boldsymbol{\Sigma} \, \boldsymbol{\mu}}{(\boldsymbol{\mu}^T \, \boldsymbol{\mu})^2}}. \tag{3}$$

Substituting $\boldsymbol{\mu}^{dem}(t)$ and $\boldsymbol{\Sigma}^{dem}(t)$ for $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ in equation 3, respectively, we get a measure of demand variability $\boldsymbol{\gamma}^{dem}(t)$. Similarly, substituting $\boldsymbol{\mu}^{sys}(n)$ and $\boldsymbol{\Sigma}^{sys}(n)$ yields a measure of system variability $\boldsymbol{\gamma}^{sys}(n)$.

B. Optimization Problem

Consequently, ABP solves a variation of optimization problem 2. Namely, given an arriving LE with demand $\mathbf{d}$ at time t and an observed relative demand measure of variability $\boldsymbol{\gamma}^{dem}(t)$, find an assignment of a PE p_n which

solves

$$\min_{n} \left(\boldsymbol{\gamma}^{sys}(n) - \boldsymbol{\gamma}^{dem}(t)\right)^2, \quad s.t.\ \psi(n; \mathbf{d}, \mathbf{a}) \geq \mathbf{0}, \quad (4)$$

$n = 1, 2, \cdots, N$.

Again, given the combinatorial complexity of solving problem 4, we employ the Biasing Sampling Algorithm (BSA) described in [3].

C. ABP Algorithm

Following is a high level description of the ABP algorithm, which is executed at the time an LE request arrives to the system. Let α be the smoothing factor parameter. A value of $\alpha = 0.001$ results in statistics being collected over 1000 samples.

1) Update statistics of relative demand (average vector and covariance matrix) by including demand $\mathbf{d}$ of incoming LE request
   a) $\nu_r \leftarrow (1-\alpha) * \nu_r + \alpha * d_r/\bar{c}_r,\ r = 1, 2, \cdots, R$
   b) $\nu_{r1,r2} \leftarrow (1-\alpha) * \nu_{r1,r2} + \alpha * (d_{r1} * d_{r2})/\bar{c}_r^2,\ r1, r2 = 1, 2, \cdots, R$
   c) $\upsilon_{r1,r2} \leftarrow \nu_{r1,r2} - \nu_{r1} * \nu_{r2},\ r1, r2 = 1, 2, \cdots, R$
   d) Compute $\boldsymbol{\mu}^{dem}(t)$, $\boldsymbol{\Sigma}^{dem}(t)$, and $\boldsymbol{\gamma}^{dem}(t)$ using equation 3
2) Collect cluster usage data
   a) Get resource utilization $\mu_{n,r},\ n = 1, 2, \cdots, N, r = 1, 2, \cdots, R$
   b) Compute $\boldsymbol{\mu}^{sys}(n)$, $\boldsymbol{\Sigma}^{sys}(n)$, and $\boldsymbol{\gamma}^{sys}(n)$ using equation 3
3) Construct $objectiveFunction$ given by equation 4
4) $n, feasible \leftarrow BSAalgorithm(objectiveFunction)$
5) return n, If feasible, and failure otherwise

Step 1 is $O(R^2)$ due to the computation of the covariance matrix and the coefficient of variation. Step 2 is similar to step 1, but repeated N times, hence is $O(NR^2)$. Given that we are placing a single LE, the BSA algorithm is $O(NR)$ [3]. Therefore, the ABP algorithm is $O(NR^2)$.

IV. Experimental Results

In this section we demonstrate the placement quality of the ABP algorithm, as contrasted to policies such as $Pack$, $Spread$, and $XBalance$, in the face of a mixed and varied workload, in terms of variation in resource demand. We conducted simulation experiments to mimic a Kubernetes cluster, hence an LE corresponds to a pod. The simulator, the placement policies, and the BSA algorithm are coded in C.

The nodes are homogeneous in terms of their resource capacities. We consider three types of resources, i.e. $\mathcal{R} = \{\text{CPU, Memory, GPU}\}$ and $R = 3$. The GPU resource is the scarce resource, hence needs to be packed. Consequently, for both $Pack$ and $Spread$ policies we select the prime resource to be the GPU ($r = 3$), i.e. we use $Pack_3$ and $Spread_3$, respectively. As for the $XBalance$ policy we use the weight vector $\mathbf{w} = [1, 0, -2]$, i.e. balance on CPU and pack on GPU with twice as much weight.

Placement requests are for single pods. We present results for using four policies: $Pack$, $Spread$, $XBalance$, and ABP. We generate Poisson arrivals with exponential lifetimes, for a total of 4,000 requests. The system starts idle and statistics are collected after a warm up period of 60 requests.

Consider a cluster with 32 homogeneous nodes comprising three resources: CPU, Memory, and GPU. The resource capacity of a node is 32, 256, and 4, respectively. The workload consists of a stream of single pods, each having one of three types: A, B, and C. The resource demand and offered load mix of the pod types are given in table I. Load is measured as CPU utilization.

	Type	A	B	C
Demand	CPU	2	8	16
	Memory	24	32	96
	GPU	0	2	4
Load	CPU	0.15	0.20	0.20

Table I
WORKLOAD MIXED RESOURCE DEMAND.

There are three phases in the experiment: I, II, and III. During phase I, the workload consists of only type A pods. During phase II, a mix of type A and type B pods make up the workload. And, during phase III, all three types make up the workload. A total of 4,000 pods arrived to the system, 666 of which during phase I, 1,334 of which during phase II, and the remainder during phase III.

The average utilization of the cluster for the three resources during the three phases is shown in figure 1. As type A pods have no GPU demand, the GPU utilization is zero during phase I. Further, the CPU utilization is about 0.15, which is the offer load of type A. During phase II, the CPU utilization is about 0.35, which is the sum of the offered load for type A and B. And, a total CPU utilization of about 0.55 is observed during phase III. Note that the GPU had the highest (bottleneck) utilization during phase III at about 0.80.

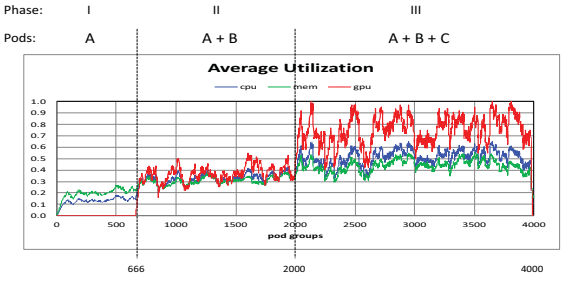


Figure 1. Load trajectory.

A summary of the results for the four policies is provided

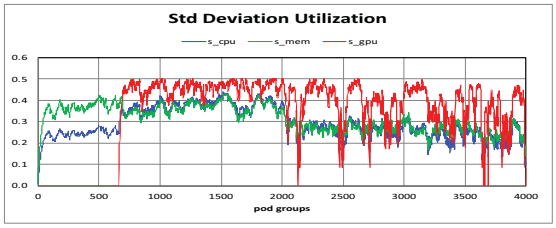
in table II, measured over the entire experiment. We are mainly concerned with three measures: (1) the rejection probability, which signifies the failure of the policy in placing pods due to lack of resources, (2) the average utilization, which is the accepted load on the system for the various resources, and (3) the standard deviation of the utilization of resources across the cluster. The latter is mainly a measure of balance. Zero deviation means complete load balance, whereas large deviation indicates uneven load distribution across the cluster, hence tendency to pack.

		PACK	SPREAD	XBAL	ABP
%	Reject	0.46	4.72	0.43	0.33
Average	CPU	0.39	0.37	0.39	0.39
	Memory	0.36	0.34	0.36	0.36
	GPU	0.50	0.46	0.50	0.50
StDev	CPU	0.29	0.18	0.14	0.22
	Memory	0.31	0.23	0.12	0.22
	GPU	0.35	0.21	0.35	0.32

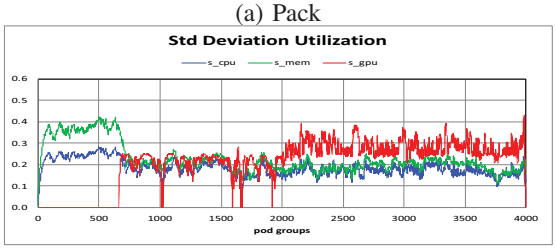
Table II
RESULTS SUMMARY.

The $Spread$ policy resulted in the highest rejection rate of 4.72%, whereas $Pack$ and $XBalance$ had a low rejection rate, with ABP slightly lower at just 0.33%. The resulting fragmentation due to the $Spread$ policy caused many rejections. Consequently, the accepted load was less than the offered load, hence a lower average utilization compared to the other policies. The measure that was significantly different among the policies was the standard deviation of utilization across the cluster. Naturally, the $Pack$ policy resulted in the largest StDev for GPU. Further, the $Spread$ policy, by design, had the smallest StDev for GPU. The $XBalance$ policy is tailored to balance the CPU and unbalance the GPU. Thus, it resulted in the lowest StDev for CPU and largest for GPU, equal to that of $Pack$. ABP on the other hand, is not performing straight packing nor spreading. Rather, it adjusts its scheme based on the observed variation in the workload demand. The result was a rather high StDev for GPU (0.32, compared to 0.35 when packing), yet a small StDev for CPU (0.22, compared to 0.18 when spreading). In other words, ABP did not go the extreme packing on GPU, while maintaining a decent balance on CPU and Memory.

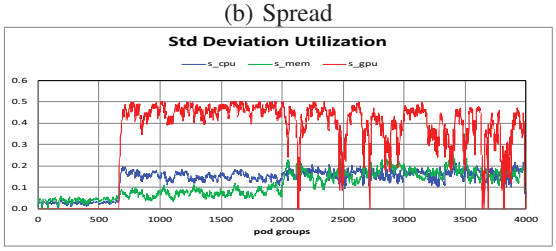
The standard deviation measure of utilization across the cluster during the experiment for the four policies is depicted in figure 2. It is clear that $Pack$ results in high variation, i.e. imbalance, whereas $Spread$ results in an opposite behavior. $XBalance$, by its objective, maximizes the variation for the GPU and minimizes it for the CPU. As for ABP, it resulted in moderate variation for all resource types, as dictated by the variation in the demand. Note that during phase I, ABP acted like $Spread$ due to the homogeneity of the workload. During phase II, ABP adapted to a behavior closer to $XBalance$, yet not too extreme. And, during phase III, ABP was closer to $Pack$.



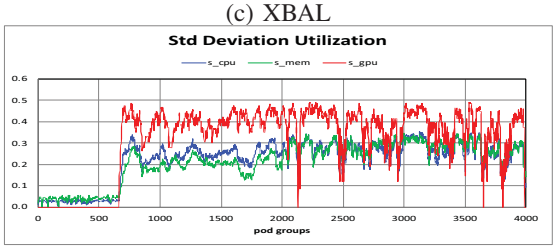
(a) Pack



(b) Spread



(c) XBAL



(d) ABP

Figure 2. Standard deviation of cluster utilization.

Table III summarizes the distribution of GPU demand of the rejected pods for the four policies. Note that the GPU node capacity is 4, hence a pod with a GPU demand of 4 takes the whole node, i.e. no other pod with nonzero GPU demand can fit on the node. Both $Pack$ and $Spread$ rejected the large (high GPU demand) pods. For $XBalance$ and ABP, the rejected pods were mostly large, but some were

GPU demand	PACK	SPREAD	XBAL	ABP
0	0	0	0	0
1	0	0	0	0
2	0	0	5	2
3	0	0	0	0
4	18	186	12	11
Total	18	186	17	13

Table III
RELATION BETWEEN REJECTIONS AND GPU DEMAND.

medium sized.

Assessing the load distribution in the cluster, resulting from the four policies, we took a snapshot of the cluster at the placement time of pod 3961, at which the average GPU utilization was about 0.70. We collected resource utilization on all nodes and ordered them in increasing order of resource utilization (GPU, followed by CPU, then Memory). The results are illustrated in figure 3. The $Pack$ policy partitioned the nodes, as far as the GPU resource is concerned, into three partitions: idle, half utilized, and full utilized. The middle partition was rather small, as expected. Further, this GPU-based partition disregarded the load distribution on the other two resources: CPU and Memory. The $Spread$ policy resulted in a large middle partition and a very small first partition, consisting of only two nodes, hence raising the chance of rejection in case more than two pods with 4 GPU demand arrive in sequence. The $XBalance$ policy resulted in partitions that are between $Pack$ and $Spread$, yet spreading the CPU (and Memory) load across the cluster. The behavior of the ABP policy was striking. In particular, it picked up the variation in the workload demand, as well as the correlation among the various resource types, resulting in a first partition which is ready to place pods with large demand of all resources. Also, note that the middle partition was rather smaller, whereas the last partition was large with positively correlated demand on GPU and the other resources.

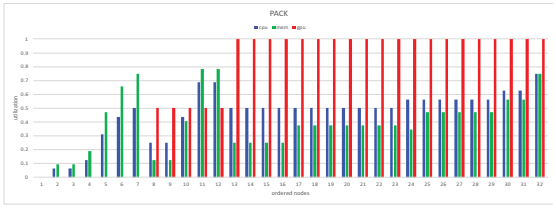
Let us consider the values of the coefficient of variation for the workload demand and the system for the four policies. Again, the data is collected at time t, taken to be the arrival time of pod 3961. Firstly, we present the statistics for the workload demand which should be independent of the policy employed. The values of the average $\boldsymbol{\mu}^{dem}(t)$ and covariance $\boldsymbol{\Sigma}^{dem}(t)$, as defined in section III, are given in tables IV and V, respectively. For completeness, we provide two sets of values, one calculated analytically from the demand mix given in table I, and the other observed during

	CPU	Memory	GPU
Analytic	0.271	0.198	0.500
Observed	0.300	0.218	0.566

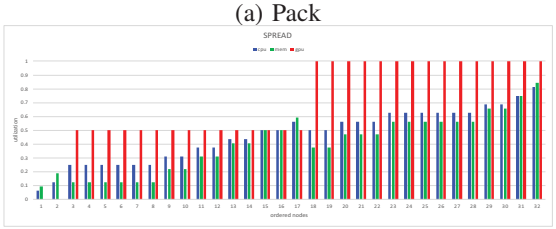
Table IV
AVERAGE OBSERVED DEMAND $\boldsymbol{\mu}^{dem}$.

		CPU	Memory	GPU
Analytic	CPU	0.032	0.021	0.073
	Memory	0.021	0.016	0.047
	GPU	0.073	0.047	0.167
Observed	CPU	0.033	0.022	0.073
	Memory	0.022	0.017	0.049
	GPU	0.073	0.049	0.166

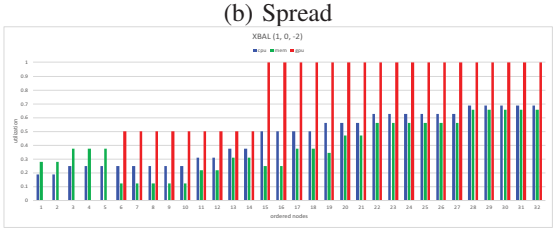
Table V
COVARIANCE MATRICES OBSERVED DEMAND $\boldsymbol{\Sigma}^{dem}$.



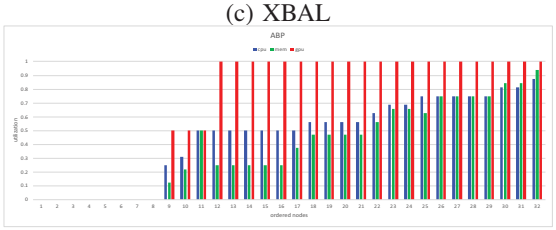
(a) Pack



(b) Spread



(c) XBAL



(d) ABP

Figure 3. Snapshot at 0.70 GPU utilization.

the simulation as described in section III, with a smoothing factor $alpha = 0.005$. The two sets were close to each other.

Secondly, we present the statistics for the system load distribution calculated when placing pod 3961, for all four policies. The values of the average $\boldsymbol{\mu}^{sys}(n)$ and covariance $\boldsymbol{\Sigma}^{sys}(n)$, as defined in section III, are given in tables VI and VII, respectively. Even though the sequence of offered load was the same for all policies, the accepted load was slightly different, hence the slight differences in the average $\boldsymbol{\mu}^{sys}(n)$. As for the covariance $\boldsymbol{\Sigma}^{sys}(n)$, the values are quite different due to the way the load is distributed in the cluster according to the various placement policies. The variance values (diagonal) relate to the standard deviation measure, which was discussed earlier. The non-diagonal values represent the correlation of placing load on the various resources of a given node. Again, the values reflect the behavior we discussed in relation the snapshot distribution depicted in figure 3.

One thing we note for the ABP policy is that $\boldsymbol{\Sigma}^{sys}(n)$

	CPU	Memory	GPU
PACK	0.459	0.396	0.703
SPREAD	0.465	0.404	0.703
XBAL	0.457	0.393	0.703
ABP	0.455	0.390	0.703

Table VI
AVERAGE RESOURCE USAGE $\boldsymbol{\mu}^{sys}$.

Policy		CPU	Memory	GPU
PACK	CPU	0.035	0.030	0.055
	Memory	0.030	0.043	0.013
	GPU	0.055	0.013	0.175
SPREAD	CPU	0.036	0.038	0.051
	Memory	0.038	0.044	0.047
	GPU	0.051	0.047	0.095
XBAL	CPU	0.033	0.029	0.061
	Memory	0.029	0.033	0.039
	GPU	0.061	0.039	0.143
ABP	CPU	0.090	0.087	0.122
	Memory	0.087	0.094	0.106
	GPU	0.122	0.106	0.191

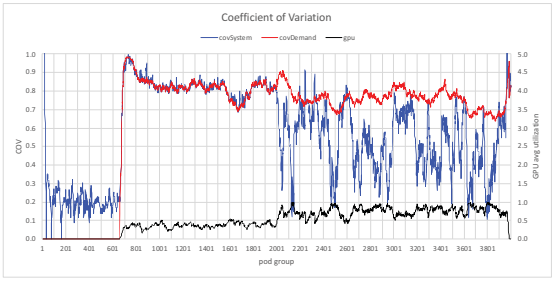
Table VII
COVARIANCE MATRICES RESOURCE USAGE $\boldsymbol{\Sigma}^{sys}$.

Analytic	Observed	PACK	SPREAD	XBAL	ABP
0.761	0.679	0.456	0.422	0.457	0.636

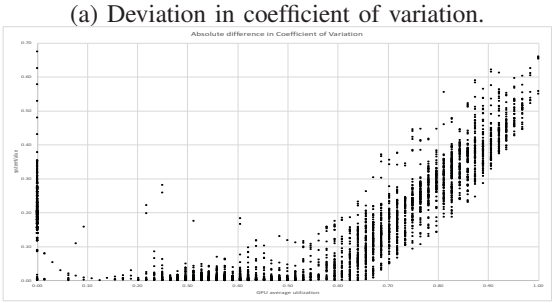
Table VIII
COEFFICIENT OF VARIATION $\boldsymbol{\gamma}$.

and $\boldsymbol{\Sigma}^{dem}(t)$, are different when considering element by element. However, the (scalar) coefficient of variation measures are quite close as shown in table VIII , where we provide the values of $\boldsymbol{\gamma}$. In particular, two sets of $\boldsymbol{\gamma}^{dem}(t)$, for analytic and observed, are given. The observed value reflects the mix observed in recent time, up to time t. Also, the values of $\boldsymbol{\gamma}^{sys}(n)$ for the four policies, are given. Note that the values were comparable for the $Pack$, $Spread$, and $XBalance$ policies, yet are far from the observed demand value, 0.679. In contrast, the ABP resulted in a value of 0.636 which is close to the demand value, as per the target of the optimization problem given by equation 4.

Now, we examine the values of $\boldsymbol{\gamma}^{dem}(t)$ and $\boldsymbol{\gamma}^{sys}(n)$ during the experiment with the ABP policy, rather than a single point, which are depicted in figure 4(a). In addition, the average GPU utilization in the cluster is provided as a reference. Further, we plot the absolute difference $\mid \boldsymbol{\gamma}^{dem}(t) - \boldsymbol{\gamma}^{sys}(n) \mid$ as a function of the GPU utilization in figure 4(b). During phase I, the non-GPU workload is homogeneous, i.e. $\boldsymbol{\gamma}^{dem}(t) = 0$, hence there is no room to bring $\boldsymbol{\gamma}^{sys}(n)$ down to zero other than attempting to spread the load. This is reflected in figure 4(b) as a collection of points on the y-axis at zero GPU utilization. During phase II, the placement engine had some room to really match the covariance values, as the GPU utilization was less than 0.50. Note that the error was small in that range. As for phase III,



(a) Deviation in coefficient of variation.



(b) Deviation as a function of utilization.

Figure 4. Error measure.

the high utilization made it harder to achieve the goal of zero difference in coefficient of variation. Nevertheless, the error was kept relatively small, yet it increases linearly with the GPU utilization.

REFERENCES

[1] Kubernetes: Production-grade container orchestration. [Online]. Available: https://kubernetes.io/

[2] M. C. S. Filho, C. C. Monteiro, P. R. Inácio, and M. M. Freire, "Approaches for optimizing virtual machine placement and migration in cloud environments: A survey," *Journal of Parallel and Distributed Computing*, vol. 111, pp. 222 – 250, 2018.

[3] A. N. Tantawi, "Solution biasing for optimized cloud workload placement," in *Autonomic Computing (ICAC), 2016 IEEE 13th International Conference on*, July 2016.

[4] X. Wei, H. Li, K. Yang, and L. Zou, "Topology-aware partial virtual cluster mapping algorithm on shared distributed infrastructures," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 10, pp. 2721–2730, Oct 2014.

[5] H. I. Christensen, A. Khan, S. Pokutta, and P. Tetali, "Approximation and online algorithms for multidimensional bin packing: A survey," *Computer Science Review*, vol. 24, pp. 63 – 79, 2017.

[6] M. C. Cohen, P. Keller, V. Mirrokni, and M. Zadimoghadddam, "Overcommitment in cloud services bin packing with chance constraints," *SIGMETRICS Perform. Eval. Rev.*, vol. 45, no. 1, pp. 7–7, June 2017.

[7] S. Aerts, G. Haesbroeck, and C. Ruwet, "Multivariate coefficients of variation: Comparison and influence functions," *Journal of Multivariate Analysis*, vol. 142, pp. 183 – 198, 2015.