Multi-resource Schedulable Unit for Adaptive Application-driven Unified Resource Management in Data Centers

David M. Gutierrez-Estevez
Huawei Technologies
Santa Clara, CA, USA
Email: david.gutierrez.estevez@huawei.com

Min Luo
Huawei Technologies
Santa Clara, CA, USA
Email: min.ch.luo@huawei.com

***Abstract*—Applications in modern data centers have a wide variety of resource requirements along the four main dimensions of computing, memory, storage, and networking. Data centers must manage these resources separately for each dimension, resulting in highly inefficient allocation of precious resources or even disastrous schemes that contribute to low utilization or over-provisioning of resources. However, concerted efforts to jointly optimize all types of resources in the same framework appears insurmountable due to an exponentially increasing complexity linked to the thousands of fine-grained resources such as CPUs, memory blocks, disk blocks, switches, etc., which results in an astronomically large number of possible resource combinations. In this paper, we present a novel multi-resource scheduling approach that keeps the complexity to a minimum while it provides efficient resource utilization by keeping a sufficient level of granularity. Our scheme is based on the idea of defining a new finest-grain schedulable unit called multi-resource schedulable unit (MRSU) that will be used by the scheduler to allocate resources to applications in the data center. The problem of MRSU-based scheduling is modeled with an optimization problem and several solutions are proposed. Our results show a significant performance improvement of our technique over conventional static schedulers based on virtual machines (VMs) in terms of saved over-allocation and application satisfaction.**

***Keywords*—Cloud computing, unified resource management, application driven, adaptive optimization, data centers.**

I. Introduction

A revolution in the information technologies (IT) has been caused by the blossoming of the field of cloud computing in the recent years [1], [2], [3]. Motivated by the fast development of processing, storage, and networking technologies, all types of computing resources have become more ubiquitously available than ever. Hence the very promising business idea of cloud computing of providing those resources as a pay-per-use service in an on-demand fashion. Central to the concept of cloud computing is the data center, the facility hosting the massive amount of compute, storage, and networking resources that enables the execution of very large-scale computations. To serve the numerous dynamic applications demanded by the tenants, a key challenge to the successful utilization of data centers is the ability to provide both efficient and effective resource management and orchestration of the complete infrastructure [4].

Among the different tasks associated with the resource management functionality in a data center, the scheduling of multi-dimensional resources across the multiple servers of a data center is a key system component that deserves special attention [5]. Two major shortcomings of current solutions are resource fragmentation and non-optimal allocation of resources that causes both *over-allocation* and *under-allocation* of resources. Firstly, most current data center resource schedulers manage resources separately for each type of resource, fragmenting the tight multi-resource operation and constraining their ability to pack tasks [6]. Schedulers usually define slots based on either only the amount of computing resources (CPU cores) or the amount of computing resources and memory, but the rest of resources are commonly ignored [5], [7]. A prominent case is the recent YARN scheduler [8], which divides computing and memory resources into slots, and offers the slots to the applications using a fair share policy. Secondly, resources are usually allocated based on application peak demand, which causes an excess of resources in high-priority applications (over-allocation) and resource starvation in low-priority applications (under-allocation). This is caused by the fact that workloads suffer from a high dynamism that may greatly change the resource requirements over its time of execution [4]. This results in non-optimal allocation of precious resources or even disastrous schemes that contribute to low utilization or over-provisioning of resources. Back to the YARN case, its scheduler uses *containers*, a logical bundle of computing and memory resources. However, it currently does not support dynamic scaling, i.e., the resource management logic assumes that the resources allocated to a container are fixed during the lifetime of it. The only way to scale it is to release it and allocate a new container with the expected size [9].

Nevertheless, concerted efforts to jointly optimize all types of resources in the same framework appears insurmountable due to an exponentially increasing complexity. In modern data centers, there are hundreds of thousands of fine-grained resources (e.g., CPUs, memory blocks, storage devices, switches, etc.) resulting in an astronomically high number of possible resource combinations. To obtain scalable data center architectures, the scheduler needs to be properly designed so that complexity is kept to a minimum while sufficient granularity is maintained and overall system performance and effective resource utilization is still satisfactory.

In this paper, we introduce the concept of multi-resource

schedulable unit (MRSU) to address current schedulers' shortcomings of peak-demand-based over-allocation and high complexity. An MRSU is a combination of resources of different nature, namely computing, memory, storage, and networking, that represent the **minimum** amount of resources that will be allocated to any application in one particular data center. Hence, an MRSU is also the finest-grain scheduling unit of the system so that only discrete amounts of this resource tuple can be allocated to the tenants' applications. Because of its multi-resource nature, a scheduling system based on MRSU is much more suited for data centers where resources are software defined and disaggregated, thus enabling significant increase in resource utilization and reduction of management complexity and cost. Moreover, the definition of this resource tuple (i.e., how much amount of each resource there is in one unit) is adaptive and changes based on the workload characteristics but avoids the high complexity of finer-grained schedulers that need to allocate resources for each dimension independently. To get a more accurate idea of the amount of complexity that needs to be handled, let's consider a typical single-site large-scale data center with 10,000 servers. Each server may have 10 CPUs with up to 10 cores each. There should be a RAM unit per CPU, and the number of storage devices or array of devices may range from 1,000 to 100,000. So the complexity to manage the resources of this typical data center independently without considering network resources would amount to 10^6 cores$\times 10^4/10^5$ RAM Units$\times 10^3/10^5$ storage devices $= 10^{13}/10^{16}$ units to manage per scheduling cycle. Furthermore, adding possible multi-paths for connecting such a device network, could easily multiply by 10 times that amount, and a distributed site could add another 10x. In summary, an insurmountable amount of complexity.

The remainder of this paper is organized as follows. Section II thoroughly describes the concept of MRSU along with the coupled joint profiling concept and a discussion on the time validity of an MRSU definition. In Section III, we formulate the MRSU scheduling problem where both an optimal MRSU needs to be defined as well as the number of MRSUs per application. We also provide an algorithmic solution of low complexity. Section IV presents performance evaluation results, and conclusions are shown in Section V.

II. Multi-resource Schedulable Unit (MRSU)-based Scheduler

In this section, we describe the fundamentals of the scheduling principle that defines our novel solution. First, Section II-A discusses the specifics of the MRSU definition and some of its main features. Second, in Section II-B we explain the concept of joint profiling and its relation to the time validity of the MRSU definition followed by a discussion on the selection of such parameter.

A. MRSU concept

Enabling adaptive scheduling is one of the major motivations behind the use of MRSU. Since current application patterns present a complex behavior characterized by time variance, location dependence, and data dependence, adaptive scheduling has become an essential tool for optimized resource management in a data center. With adaptive scheduling, decisions can be made based on the characteristics of the tenants' applications rather than allocating a fixed number of each type of resource sufficient to handle some expected resource requirement. The relevant characteristics of the application may include its behavior over time, its location, Oftentimes it is peak demand what is used as expected requirements to prevent applications from not meeting their QoS. This type of allocation is static and does not satisfy demands in a dynamic fashion as it would be desired so that allocations could be matched with application needs at critical time instants. Moreover, an application-driven adaptive system may serve more applications with the same amount of infrastructure as static allocations since application processes can be dynamically squeezed into a schedule using idle resources.

The above-mentioned shortcomings of current schedulers lead us to propose a new approach called multi-resource schedulable unit (MRSU). An MRSU is a minimum multi-dimensional schedulable unit for a single multi-tenant data center comprising a specified amount of computing, memory, storage, and networking resources that are jointly schedulable and serve a minimum scale application with required quality-of-service (QoS). A minimum combination of all types of resources is selected with the objective that all the tenants demands can be expressed as multiples of the MRSU. The reason for this system constrain is to greatly simplify the design and operation of the scheduler, which would only need to allocate a discrete number of MRSUs for each application, hence accounting for the resources needed in the four dimensions.

Key to the performance of this novel scheduling technique is the definition of the MRSU, i.e., how many resources form each of the dimensions of the MRSU tuple. More formally, the definition of an MRSU tuple Ψ is given by

$$\Psi = \{x, y, z, w\}, \tag{1}$$

where x is the amount of computing resources measured in number of CPU cores, y is the amount of memory resources measured in blocks of RAM (usually in GBs), z is the amount of disk storage resources measured in blocks of GBs, and w is the amount of networking resources measured in network bandwidth. Regarding network bandwidth, it is important to mention that we don't neglect the fact that multiple paths can be followed by the different flows of a single application to reach a destination server. Instead, we consider the network bandwidth requirements and offerings as belonging to a limited set of most critical paths connecting the source and destination nodes within the data center.

The derivation of an MRSU for different data centers with different tenants and applications will yield different MRSU definitions. Moreover, the definition will most likely need to be dynamically changed to adapt a suitable MRSU configuration to time-varying, location, and data dependent application patterns. Therefore, the resource allocation process based on MRSU has two phases: i) The derivation of a definition for an MRSU based on the data center load, i.e., the determination of how many resources form an MRSU in each independent dimension, and ii) the calculation of the number of MRSUs that will be allocated to each application given the previous definition. In Section III, an optimization problem modeling this complex two-phase problem is presented. However, both steps are large, NP hard discrete problems requiring a much

less complex solution.

Figure 1 shows a complete schematic sample diagram of the resource allocation operation in a data center using the MRSU principle. As shown in Fig. 1(a), the data center has some capacity of computation availability for processing - usually central processing units (CPUs), some capacity of RAM memory for storage during processing, some capacity for long term memory storage, and some network capacity for data communication. A plurality of tenants (in our example tenants 1, 2, and 3 with one application to be processed each) require some portion of each of the resources. Each tenant may be requiring different amounts of resources in different ratios from other tenants.

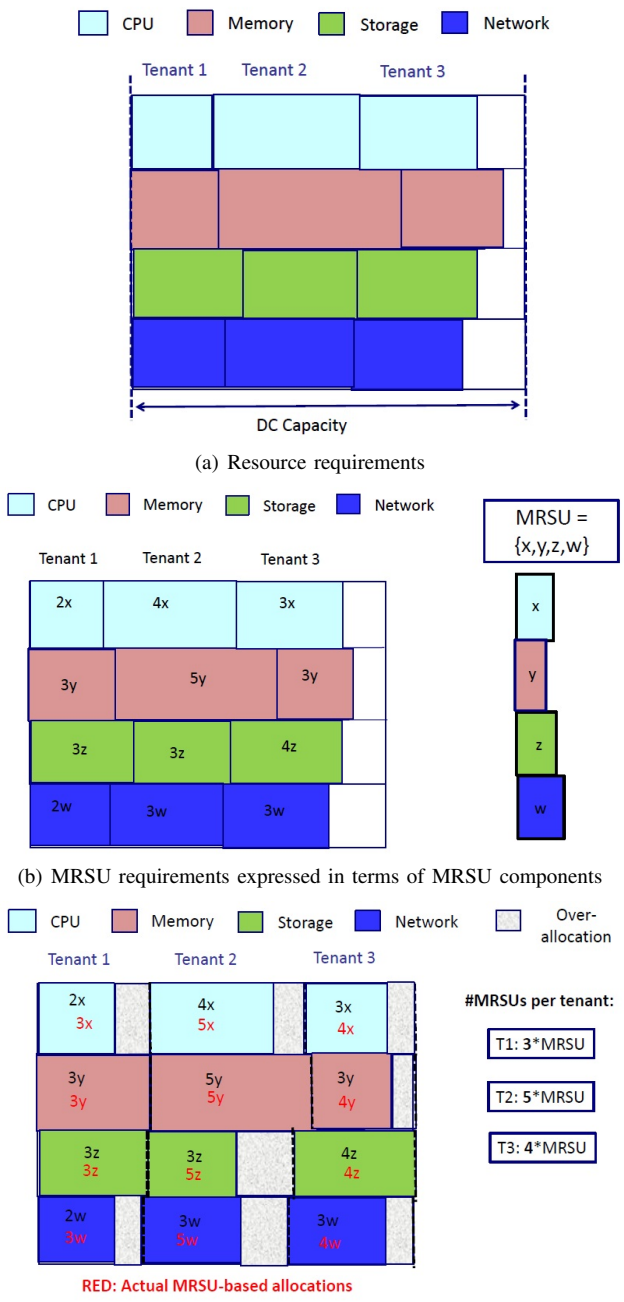
Figure 1(b) shows the MRSU tuple that will be used for the set of applications while the current MRSU definition holds. As shown, each tenant requires different amounts for different resources. In our example, the MRSU consists of x units of CPU cycles, y units of cache space, z units of storage space, and w units of network bandwidth. Then, the requirements of each application can be expressed in terms of the MRSU per-dimension unit obtained. Finally, Fig. 1(c) shows the actual allocations given the fact that a natural number of MRSUs has to be allocated to each application. As it can be seen in the figure, the constraints of MRSU-based allocation creates some over-allocation in the system due to the heterogeneity of the resource demands in the different dimensions. Obtaining the optimal MRSU definition and allocation will allow to minimize this over-allocation while keeping the significant benefits of complexity reduction. A four dimensional vector can be employed as four resource types are employed. Furthermore, the allocation in terms of number of MRSUs can be represented by a vector of dimensionality equal to the number of applications. The actual MRSU allocation for each tenant should meet or exceed the requirements of that tenant for a given resource. The details of the MRSU computation are discussed in Section III.

Independent-dimension MRSU: Alternatively to the definition presented above, an MRSU may also be defined independently for each dimension in such a way that resources for each dimension are allocated independently of the rest of dimensions in the MRSU. For example, if a application requests the following tuple of resources expressed in terms of the MRSU $3x, 4y, 2z, w$, the allocation in each dimension is independent of each other and only limited by the rest of applications and the data center capacity. This approach logically reduces over-allocation due to MRSU constraints but increases the complexity of the scheme as allocations are to be calculated and executed independently for each dimension.

This paper considers the two definitions of MRSU introduced in the above paragraphs, and Section IV shows how their performance compares between them and with respect to other schemes.

B. Joint Profiling

Processor allocation profiles, memory profiles, and/or any other resource profiles can be combined in a joint resource allocation profile for the data center. The derivation of a single MRSU for the multi-tenant data center combines the various resource profiles into the joint profile of the set of



(a) Resource requirements

(b) MRSU requirements expressed in terms of MRSU components

(c) MRSU allocations and over-allocation

Fig. 1. MRSU utilization example

applications being run by each tenant. Application profiles should be combined across applications and across tenants so that a joint profile is utilized to derive data center-wide MRSU. Once again, we see that the resulting MRSU definition is adaptive and tunable for each data center based on the workload dynamics of the hosted tenants.

The use of joint application profiles across tenants, applications, and the data center is advantageous. The key advantage of this technique is its ability to significantly increase resource utilization, hence being able to satisfy more applications' needs for resources, which in turn enables more applications to be easily run with the same infrastructure. Furthermore,

the consideration of joint application profiles dependency on location, time, data distribution and QoS requirements differentiates the proposed approach from the virtual machine (VM) based market place. Pre-allocating resources by peak demands for each application/tenant results in wasted resources over most time. By joint profiling and global resource allocation optimization, resource requirement variation over time can be utilized to select a proper MRSU that increases overall resource utilization, and/or supports an ability to serve additional applications with the same infrastructure. This also shows that our approach enables mostly autonomous resource allocation and management, which significantly reduces management complexity and cost.

C. Discussion on the Time Validity of MRSU (T)

The MRSU must be derived and kept for certain period of time during which the scheduler will be able to just allocate MRSU units defined in certain way. It is an interesting discussion point the observation that the optimal definition of an MRSU may differ at each point depending on the characteristics of the data center load. For example, if the workload requirements at certain time of the day are very dynamic, it may be optimal to re-calculate the MRSU with more frequency to better adapt to the load. This requirement may be relaxed if the load characteristics are very homogeneous and do not vary greatly over time. Hence, the time validity of an MRSU could be set to any time interval ranging from minutes to hours, and the frequency used to recalculate its value need not be a constant time duration either. Although out of the scope of this paper, joint profiles as described above may be a good tool to estimate the optimal T for a certain combined load as well as the time duration of that calculated frequency.

III. Problem Formulation and Proposed Solution

A. The Optimization Problem Formulation

An optimization problem can be used to model the MRSU scheduling problem, where both the MRSU definition vector for every time interval T must be obtained as well as the number of MRSUs allocated to each application. The formulation tries to minimize over-allocation of resources over each time interval T, while guaranteeing that all capacity, fairness, prioritization, and resource requirement constraints are met. The optimization problem employs an objective function as shown in Eq. (2). Specifically, an optimal allocation is sought in terms of the MRSU definition vector Ψ and the allocation vector a containing the number of MRSUs per user based on the following objectives: i) over-allocation should be minimized, ii) minimum/maximum fairness should be guaranteed, and iii) priorities should be respected in the allocation. The absolute value in Eq. (2) simultaneously accounts for over-allocation and user satisfaction: We would like the difference of allocated resources and requirements as close to zero as possible. In addition, priorities can be respected by weighting the former difference with the priority value of each application. The constraints of the problem are simple. First, constraint (3) accounts for the capacity of the data center in each dimension, i.e., the sum of the allocations in each dimension cannot exceed this value; second, constraint (4) guarantees that the number of MRSUs allocated to each application at each time instant is a positive integer; and finally, constraint (5) guarantees that the MRSU is a d-dimensional vector whose components are all integer numbers greater than zero.

$$\min_{A,\mathbf{\Psi}} \max \sum_{n=1}^{N} \sum_{d=1}^{D} \sum_{t=1}^{T} \omega_n \left| \frac{\Psi_d}{T} a_n^t - R_{nd}^t \right| \tag{2}$$

Subject to:

$$\Psi_d \sum_{n=1}^{N} a_n^t \leq C_d \quad \forall d, \quad \forall t \tag{3}$$

$$A = \{a_n^t\}_{n=1,\cdots,N}^{t=1,\cdots,T} \quad a_n^t \in \mathbb{N} \quad \forall n, \quad \forall t \tag{4}$$

$$\mathbf{\Psi} \in (\mathbb{N}^+)^D \tag{5}$$

In the above formulation, N is the number of applications, D is a number of resource dimensions, T is a length of interval for which the MRSU definition is valid, C_d is the data center capacity in dimension d, R_{nd}^t is the peak resource requirements for application n in dimension d at time instant t, A is the time allocation matrix indicating a number of MRSUs allocated to each application at each time instant, and ω_n is the priority of application n where $0 \leq \omega_n \leq 1$.

The optimization problem is highly complex in its original form because of the min-max objective function and the fact that we need to determine both the components of Ψ and the number of MRSUs that need to be allocated to each application at each time instant. In the following we propose a low-complexity algorithmic solution for the above optimization problem based on a modified version of the well-known weighted fair queuing scheduling method.

B. "Weighted Fair MRSU" Algorithm

In this section, we present a heuristic method to solve the problem of the MRSU allocation. An algorithm has been designed with the same objective as the optimization problem: Over-allocation needs to be minimized, resource demands should be met as much as possible, and users should be served based on their priorities. The algorithm has two parts: The first one calculates the composition of the MRSU vector Ψ as well as the exact number of MRSUs available for allocation, and the second one obtains the number of MRSUs per application $\mathbf{a}$. The second part of the algorithm is an adaptation of weighted fair queuing (WFQ) scheduling policy [10] to the specifics of MRSU: Units of MRSU are allocated in rounds proportionally to the priorities, and unnecessary allocated resources to applications already meeting their requirements are subtracted and reallocated to other applications not having met their resource demand following a prioritized order.

Specifically, the algorithm steps are as follows:

- **Part I: MRSU composition and availability.** Using the previously derived validity interval T, the MRSU unit has to be obtained. We consider two different methods to obtain the different components of the MRSU:
    - *Greater Common Divisor (GCD):* For each dimension, choose the value that allows the best fitting of the resources in the provided

capacity by calculating the greatest common divisor for each dimension among the application requirements and the capacity for such specific dimension. The sought vector is given by

$$\Psi_d = \text{GCD}(R_{1d}, R_{2d}, \cdots, R_{Nd}, C_d) \quad (6)$$

where the computation of the GCD of N+1 values can be efficiently performed by employing the well-known Euclid's algorithm [11] and the following recursive GCD operation property: $\text{GCD}(x, y, z) = \text{GCD}(\text{GCD}(x, y), z)$.

- *Minimum:* For each dimension, the minimum requirement is selected across all applications as the MRSU component in that particular dimension:

$$\Psi_d = \min\{R_{1d}, R_{2d}, \cdots, R_{Nd})\} \quad (7)$$

In this approach, there is no guarantee that the requirements of the applications not selected as minimum will be conveniently expressed in terms of the minimum, hence creating the risk of high over-allocation. Furthermore, resources might get also wasted because the capacity of that dimension is not taken into consideration either.

Having determined the components of the MRSU, we now need to obtain the maximum number of available MRSUs for allocation to the hosted applications. The main constraint is that the number of available MRSUs does not exceed the data center capacity in any of its dimensions. Following the standard definition of an MRSU where tuples of resources are allocated, the amount of MRSUs would be naturally given by the dimension that fits the least amount of resource blocks as given above. However, an alternative approach exists where complexity would be traded off for flexibility: Treating the MRSU dimensions independently and allocating *MRSU components* instead of full tuples. Hence, the two methods to obtain the number of available MRSUs are as follows:

- MRSU-FT: If full MRSU tuples are allocated, we denote the amount of available MRSUs with the scalar X. To guarantee that the allocation of MRSUs does not exceed the data center capacity in any of its dimensions, the method to obtain X is given by

$$X = \min\left\{\frac{C_1}{\Psi_1}, \frac{C_2}{\Psi_2}, \cdots, \frac{C_N}{\Psi_N}\right\}. \quad (8)$$

Hence, the number of available MRSUs will be determined by the lowest ratio of capacity over the corresponding MRSU component for all dimensions, which will unavoidably introduce some level of over-allocation in the system.

- MRSU-ID: If MRSU dimensions can be independently allocated, the number of available MRSU components is denoted by the vector $\mathbf{X}$, where each component of $\mathbf{X}$ contains the number of MRSU units in each dimension:

$$\mathbf{X} = \left\{\frac{C_1}{\Psi_1}, \frac{C_2}{\Psi_2}, \cdots, \frac{C_N}{\Psi_N}\right\}. \quad (9)$$

As mentioned above, this will introduce additional flexibility in the system at the expense of incurring in a higher complexity since the scheduler has to centrally decide the number of MRSU components allocated in each dimension for every application instead of just computing a single quantity representing the full resource tuple.

- **Part II: MRSU allocation** The second part of the algorithm determines the number of MRSUs allocated to each application. For that, priorities should be normalized amounting to a total of 1. The procedure explained below assumes that full tuples are allocated (i.e., the first of the two approaches described above). However, the modifications required for the approach based on independent MRSU components is straightforward as the only difference is that the same algorithm has to be applied for each resource dimension independently. Our proposed procedure goes as follows:
  1) The discrete number of MRSUs for each application x_n is obtained by calculating the proportional amount of MRSUs corresponding to each application according to its priority followed by a floor operation to guarantee capacity is not exceeded, i.e., $x_n = \lfloor \omega_n \cdot X \rfloor$.
  2) Then, the remaining amount of MRSUs $\hat{X}$ is calculated by computing the number of MRSUs that can be subtracted to high-priority or low-demand users who have already satisfied their demand with the previous allocation. Furthermore, the remaining MRSUs resulting of the discretization step are also added to the pool of available MRSUs.
  3) At this stage, terminate if the previous step does not yield any remaning MRSUs.
  4) Else, re-normalize priorities eliminating satisfied applications to make all unsatisfied applications' priorities amount to 1.
  5) Go back to 1).

IV. Performance Evaluation

In this section, we evaluate the performance of our proposed scheduling approach by examining two major metrics: i) over-allocation, i.e., the amount of wasted resources that are unnecessarily allocated even if other applications are suffering from not receiving as many resources as needed, and ii) the application dissatisfaction, i.e., the amount of applications that do not get the amount of resources they need. Here, it is also interesting to have a sense of how big the shortage of resources is. The benchmarking will be performed among different scheduling techniques: i) conventional static VM-based allocation ii) MRSU-based full-tuple allocation, iii) MRSU-based independent-component allocation, iv) optimal allocation, where the scheduler is capable of allocating at every time instant exactly the amount of resources needed by the

application requirements. This last option is considered as a lower bound. The evaluation is performed using simulation in MATLAB where the random number generator uses a large number of seeds to guarantee statistical significance. Synthetic workload traces are generated representing the application requirements. The workloads requirements are generated using independent uniform random distributions for each dimension. Most of the simulation results are shown against the average load of the data center, expressed as a percentage of the total capacity. This average load is hence represented by the mean of the uniform random distribution. In the rest of the section we first introduce the *misallocation coefficient*, a metric designed to jointly quantize over-allocation and application dissatisfaction, and subsequently we show the simulation results.

A. Misallocation coefficient

The objective of defining a specific metric to account for the *misallocation* of the system is to capture in a single metric the effects of both allocating more and less resources than what is needed. This is clearly feasible: Intuitively, one can easily guess that what is sought is to make requirements and allocation equal to each other (or, equivalently, their difference equal to zero) to minimize the two metrics discussed above. In addition, the metric should be normalized so that it can be compared across dimensions where different measurement units are utilized (e.g., CPUs, memory blocks, network bandwidth, etc.)

Let us call Y the amount of resource allocated to a particular application in one particular dimension (computing, memory, storage or networking) and Z the requirement of that same application for that same dimension. We define the misallocation coefficient $/ro$ as follows:

$$\rho = \frac{Y - Z}{\max(Y, Z)}, \tag{10}$$

where the denominator ensures that $\rho \in [-1, 1]$. Furthermore, this metric allows to determine at a glimpse whether resources are being over-allocated or under-allocated just by observing its sign: A positive misallocation coefficient means resource over-allocation while negative misallocation coefficient implies application dissatisfaction. Figure 2 shows the snapshot of a misallocation coefficient for a sample data center with ten applications where the scheduling mechanisms are both conventional VM-based and MRSU-based. The index in the X-axis represents the coefficient for a particular dimension of a particular application in a particular time slot, where the number of dimensions in four and the simulation lasted for thirty time slots. We see the existence of over-allocation and application dissatisfaction in both cases, although MRSU outperforms VM-based in both metrics. In fact, very little application dissatisfaction can be observed in MRSU and a large number of indexes obtain the exact requested amount of resources.

B. Results

The results of the simulations are shown in this section. The misallocation coefficient metric is used to identify the amount of application dissatisfaction and over-allocation caused by the different scheduling techniques. To further clarify the results,

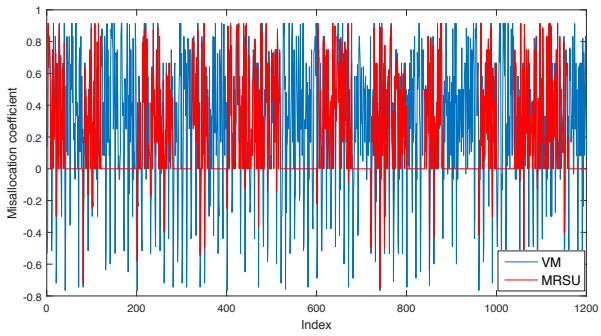
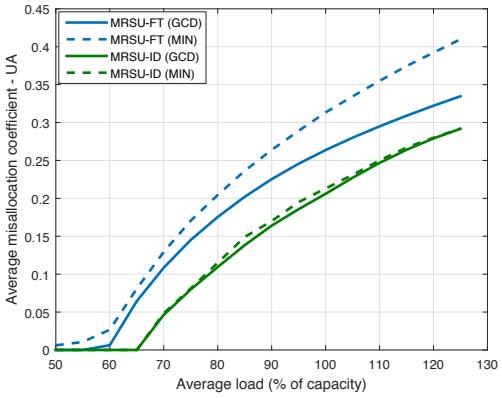


Fig. 2. Misallocation coefficient snapshot

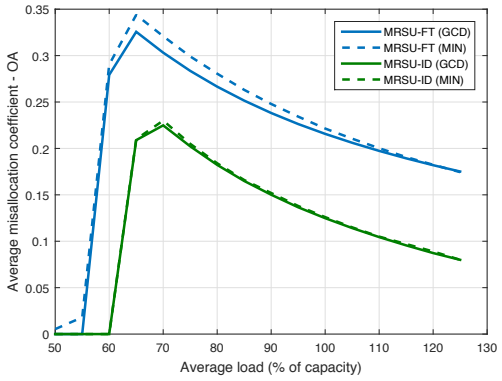
we separately show the results concerning application dissatisfaction and over-allocation as an average of the misallocation coefficient across dimensions and time slots. In both cases we show the absolute value of ρ as once identified the case of over or under-allocation, we are mostly interested in knowing the magnitude of such mismatch. We evaluate ρ for an increasing average load of the data center. The random load that arrives to the data center is characterized by its average and represented as a percentage of the total capacity of the data center.

Our first experiment is targeted at determining the best approach to compute the MRSU components from the the requirements of the different applications in each separate dimension. As explained in Section III-B, two possible computations for the set of requirements are proposed, namely GCD and MIN. Figure 3 shows the comparison of the two components of the misallocation coefficient (application dissatisfaction and over-allocation) only for the cases of MRSU computation - hence, no VM allocation is shown here. We observe that the GCD computation consistently outperforms MIN by showing a smaller misallocation coefficient for both components of ρ and both MRSU-FT and MRSU-ID cases, although the performance difference is clearly larger in the MRSU-FT case. The reason lies in the additional lack of flexibility that the MIN approach encompasses: If the number of allocated resources is constrained to be a multiple of the MRSU, selecting the minimum does not guarantee that the selected quantity could be efficiently used to express the requirements of the rest of the applications and efficiently fit the capacity of the data center. This causes the wastage of a part of the resources in high-priority applications, but more importantly, the dissatisfaction of low-priority applications that do not receive the requested resources.

Figure 4 shows the simulation results for the two cases mentioned above. In Fig. 4(a), the application dissatisfaction is shown as the *under-allocated* portion of the misallocation coefficient. For the region of interest, i.e., the part of the curve where the average load does not exceed capacity, both MRSU approaches show a smaller misallocation coefficients than the regular VM-based scheduler. Indeed, the dynamic nature of the scheduler allows to provision resources to applications of lower priorities who would otherwise suffer of a greater performance loss if the resources were statically allocated based on peak demand. Furthermore, as it is expected, the independent-dimension version of the MRSU allocation achieves lower application dissatisfaction due to the inherent flexibility to

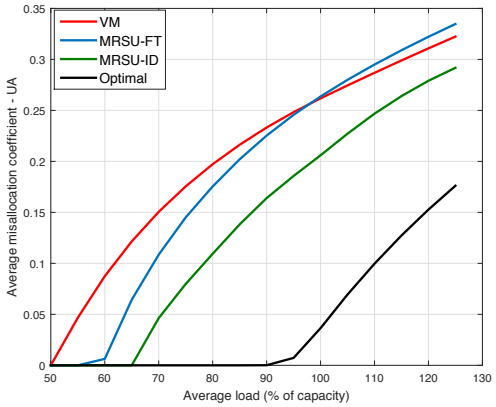


(a) application dissatisfaction portion

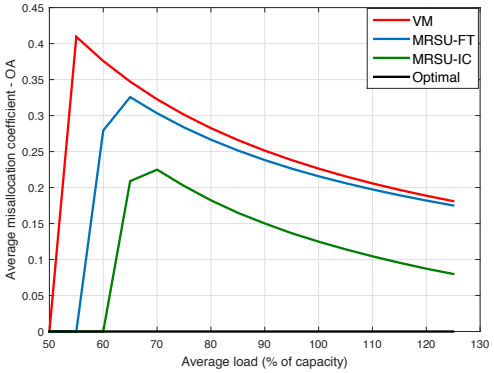


(b) Over-allocation portion

Fig. 3. Average misallocation coefficient comparison for GCD vs MIN.



(a) application dissatisfaction portion



(b) Over-allocation portion

Fig. 4. Average misallocation coefficient for different scheduling techniques.

the allocation of MRSU components instead of full tuples. It is also worth noting that the VM scheduler starts causing less application dissatisfaction than MRSU-FT after capacity is reached. As it will be shown later, this does not mean that the number of applications which meet their requirements decreases, but the amount of resources that each application is lacking to meet their requirements is larger. Finally, the optimal version of the scheduler does not feature any application dissatisfaction almost until the moment that the average load equals capacity. This lower performance bound is quite distant from the rest of scheduling schemes.

Regarding over-allocation, Fig. 4(b) shows the behavior of the *over-allocated* portion of the misallocation coefficient. Overall, the hierarchy of solutions is the same, with the optimal solution not producing any over-allocation in the system as only the required resources are provided to the applications. The shape of the VM, MRSU-FT, and MRSU-ID curves is similar: Over-allocation starts at some load point and rapidly increases achieving a peak. Then, it starts decreasing at a lower rate. The explanation is as follows: ρ accounts for over-allocation only when a application is not meeting its requirements. Hence, as soon as some application dissatisfaction exists, the over-allocation of the system will increase. The rapid growth is caused by the fact that load is still low, so it seems reasonable that the amount of excess of resources in higher-priority applications will be larger. Indeed, the curve stops increasing once there starts being less excess of resources for high-priority applications, i.e., when the load of the system increases. At this point, the over-allocation curve starts decreasing simply because there are less resources to cover the demand. Notice, however, that this over-allocation does not disappear even when the average load of the system grows as high as 125% of the capacity.

Finally, we explore the plain number of applications that do not get their resource requirements as a function of the load. This metric is independent of the misallocation coefficient but is intended to validate the results above. Figure 5 shows the results, where the same performance hierarchy is observed as in Fig 4(a). Interestingly, we observe that the number of dissatisfied application is always larger in VM-based scheduling than any MRSU version. This means that the crossing of the VM and MRSU-FT curves in Fig. 4(a) is indeed due to a greater distance from the allocated resources to the requirements in applications of low priority, but not to a greater number of applications suffering dissatisfaction.

V. Conclusions

In this paper, we have presented a novel multi-resource scheduling scheme for data centers that addresses the problem of keeping the complexity low while guaranteeing an efficient resource allocation that minimizes both over-allocation and application dissatisfaction through a highly adaptive dynamism that current techniques lack. Our scheme is based on the idea

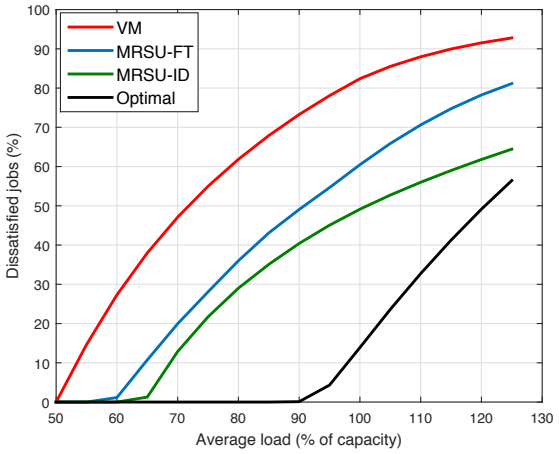


Fig. 5. Percentage of dissatisfied applications

of defining a new finest-grain schedulable unit called multi-resource schedulable unit (MRSU) that will be used by the scheduler to allocate resources to applications in the data center. Hence, the MRSU determines the minimum amount of resources in each dimension that can be allocated. We have formulated the problem of MRSU allocation as an optimization program trying to minimize over-allocation and a solution inspired in the weighted fair queuing algorithm is proposed. The performance evaluation results show a significant performance improvement of our technique over conventional schedulers based on VMs in terms of both saved over-allocation and application satisfaction.

Two lines of work will be followed in the future. The first one intends to to provide an extension of current open-source resource schedulers to support the MRSU concept. In particular, we are interested in providing an implementation for the schedulers used in Openstack and Spark. Underlying Spark there is the resource management framework Mesos, which uses the concept of resource offer that could be easily extended to support MRSU. The second line of work intends to provide a real-time joint solution to the full optimization problem without the need for a two-phase suboptimal algorithm.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of internet services and applications*, vol. 1, no. 1, pp. 7–18, 2010.

[3] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica, "Above the clouds: A berkeley view of cloud computing," *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, vol. 28, p. 13, 2009.

[4] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *Journal of Network and Systems Management*, pp. 1–53, 2014.

[5] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types." in *NSDI*, vol. 11, 2011, pp. 24–24.

[6] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014, pp. 455–466.

[7] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009, pp. 261–276.

[8] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*, 2013, p. 5.

[9] Hadoop YARN Project, "Support changing resources of an allocated container," https://issues.apache.org/jira/browse/YARN-1197, accessed: 2015-07-08.

[10] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," in *ACM SIGCOMM Computer Communication Review*, vol. 25, no. 4, 1995, pp. 231–242.

[11] Wikipedia, "Euclidean algorithm," https://en.wikipedia.org/ wiki/Euclidean_algorithm, accessed: 2015-07-08.