ABSTRACT

Providing performance predictability guarantees is increasingly important in cloud platforms, especially for data-intensive applications, for which performance depends greatly on the available rates of data transfer between the various computing/storage hosts underlying the virtualized resources assigned to the application. With the increased prevalence of brokerage services in cloud platforms, there is a need for resource management solutions that consider the brokered nature of these workloads, as well as the special demands of their intra-dependent components. In this paper, we present an offline mechanism for scheduling batches of brokered data-intensive workloads, which can be extended to an online setting. The objective of the mechanism is to decide on a packing of the workloads in a batch that minimizes the broker’s incurred costs. Moreover, considering the brokered nature of such workloads, we define a payment model that provides incentives to these workloads to be scheduled as part of a batch, which we analyze theoretically. Finally, we evaluate the proposed scheduling algorithm, and exemplify the fairness of the payment model in practical settings via trace-based experiments.

1. INTRODUCTION

Data-intensive applications, such as MapReduce [16], and Message Passing Interface (MPI) [38] applications, devote most of their processing time to data transfer and manipulation, and their execution time depends mainly on the size of the data to be processed, and the characteristics of the network over which this data is transferred [34]. Resource virtualization has enabled data center providers to offer such applications virtually isolated computing resources for a fraction of the cost of the actual physical hardware. However, these virtual resources share the data center’s network infrastructure, the fabric, and thus allow all applications to access the fabric in an uncontrolled and opportunistic manner, resulting in unpredictable network performance, which in turn affects the execution times of such data-intensive applications, and consequently their costs [24, 28, 7].

Service brokerage in a cloud computing setting, as that presented in [8], allows brokers to act as intermediaries between the resources offered by the data center providers, such as Amazon EC2 [2], and their consumers. This brokerage model creates a marketplace, in which the resources of existing cloud platforms are used to provide predictability guarantee services tailored according to the customer’s need, at a competitive price.

Figure 1 presents an architectural view of a cloud brokerage framework, in which the broker has no control over the internal management structures of neither the data center provider nor the application’s workloads. Virtual machine container/slots are rented out from data center provider(s), and an inference service is used to estimate the physical resources achievable from them. On the applications’ side, a profiling service is used to understand the behavior of each application’s internal components, generating its corresponding resource and temporal demands. The resource management service uses the information extracted by the inference and profiling services, and manages the resources rented from the data center provider(s). The objective of resource management is to pack the components of the brokered workloads into the brokered resources, such the application’s predictability is guarantees and the broker’s costs are minimized.

Resource management includes various components, of which we consider the resource allocation, scheduling, and payment modules. The resource allocation module is in
charge of mapping the components of a single profiled application to the appropriate VM slots coordinated by the broker, with an objective to offer the highest quality service to the application. The scheduling module is in charge of finding the optimal schedule of execution for multiple applications arriving at the system, with an objective to maximize the utilization of resources and minimize the broker costs. Finally, the payment module is in charge of deciding on the fair distribution of the broker’s costs on the various applications.

Current schedulers do not consider the special demands of data-intensive applications, with intra-dependent components, nor do they consider the brokered nature of these workloads. In this paper, we present an offline mechanism for scheduling batches of brokered data-intensive workloads, which can be extended to an online model. The objective of the mechanism is to decide on a packing of the workloads in a batch that minimizes the broker’s incurred costs\(^1\), as well as the cost of the packing itself, as defined in [8].

Moreover, since the brokered workloads are assumed to be individually-owned and controlled by rational entities with independent utilities, there is a need to provide incentives to these workloads to be flexible with their temporal demands. This flexibility in temporal demand allows the broker to group more jobs per batch, thus achieving better packing and utilization of resources. Adopting the economical model of Vickery-Clarke-Groves mechanisms [35], we define an efficient payment model to be associated with any scheduling algorithm for data-intensive workloads. We prove the model to be incentive compatible and individually rational, with the use of optimal scheduling algorithms. Moreover, we show via experimental evaluation that the fairness of these VCG-based payments is comparable to their corresponding estimated Shapley values [13], with the advantage of reduced computation complexity.

Following the definition of an incentive-compatible scheduling and resource allocation mechanism for a batch of data-intensive workloads, associated with a fair payment model, the final contribution of this paper is the definition of an online scheduling heuristic that uses backfilling techniques [40, 39] to group workloads arriving in an online fashion into batches. The presented online scheduling mechanism allows for better packing and broker costs, and fair payment guarantees for the workloads.

Paper Outline.

In Section 2, we start with the definition of the FlexPack problem, and an analysis of its hardness. In Section 3, we present a scheduling mechanism for a batch of time-flexible data-intensive applications, which is composed of a heuristic to solve the FlexPack problem, and an efficient payment model. In Section 4, we extend the proposed batch scheduling mechanism to online settings. Finally in Section 5, we evaluate the scheduling mechanisms in offline and online settings, as well as the payment model, and we conclude the paper with conclusions and future work in Section 6.

---

\(^1\)In the context of this paper, we only consider the costs of renting the VM slots from the cloud providers.

---

2. SCHEDULING BROKERED DATA-INTENSIVE WORKLOADS

2.1 Problem Model

We define the service broker, as an entity that has access to only a subset of the VM slots provided by the data center provider. The broker is oblivious to the status of other VM slots in the data center, and the data center fabric.

An inference service groups colocated VM slots into \(m\) servers, and represents the properties of these servers by the tuple \((D, C, B)\). The distance matrix \(D : \langle m \times m \rangle\) represents the relationship between the servers, i.e., the value of \(d_{i,j}\) could represent the network distance between the servers, the delay of communication between them, or some other cost of communication metric as defined by the service provider. The matrices \(C : \langle m \times H \rangle\) and \(B : \langle m \times H \rangle\) represent the computing and bandwidth resources available at each server for all time units\(^2\) in the time epoch\(^3\).

We consider a batch of jobs (workloads) \(J\) requesting service from the broker, within which a single job \(J_i\), is defined by the tuple \((n_i, T_i, s_i, d_i)\). Each job has a hard deadline of \(d\) units from its arrival, and an execution span of \(s\) time units from when it starts. A scheduled job is allocated the resources it needs for a duration of its span, and then these resources are released for other jobs to use after the job leaves the system. Moreover, the job’s VMs once allocated resources on a server can’t be preempted, or migrated to another server.

2.2 Problem Definition

The FlexPack problem is that of finding the optimal allocation of resources to a batch of workloads, which minimizes the total cost of the broker. The total cost of the broker is defined as the cost of maintaining the VM slots required to provide predictability guarantees for the epoch’s duration. These include the VM slots utilized by the workloads, as well as the ghost slots maintained by the broker to sustain the bandwidth capacity of its server.

2.3 Problem Formulation

We formulate the optimization problem of the FlexPack problem with the decision variable \(X_{i,k,t}^{j}\), which is a boolean variable indicating whether the \(k\)’th VM of the \(i\)’th job is mapped to server \(j\) at time step \(t\) for its \(l\)’th duration step of its span.

The objective is to maximize the difference between the revenue obtained from the jobs scheduled, and their allocation cost.

Minimize:

\[
\sum_i \left( \left( R_{n_i, s_i} \sum_{j, t} X_{i, k, t}^{j} \right) - \left( \sum_{j, k, t} (X_{u, k, t}^{j} \times R_{k, t}) \times d_{j, k} \right) \right)
\]

The constraints are to guarantee that the total number of VMs, from all scheduled jobs, don’t exceed the computing

\(^2\)Time is modeled as discrete time steps.

\(^3\)A consecutive sequence of time steps. All workloads in a batch are assumed to be ready at the beginning of the considered epoch, during which all scheduling and allocation decisions are made.
execute the NCP algorithm at least
the optimal allocation of resources to jobs, we have to ex-
of its span,
have been mapped to a server, at the same time units for all
with
resources to a single job in polynomial time. In an instance
shot of the system, we can find an optimal allocation of
for the general NCP problem, this means that given a snap-
problem. Assuming we have an exact polynomial algorithm
even with the existence of an efficient solution for the NCP
becomes identical to that of the Network-Constrained Pack-
cost. Consequently, the objective of the FlexPack problem
job at any time within its deadline won’t affect its packing
do not change throughout the time epoch, and starting the
job, we guarantee that the parameters of the service broker
servers before its deadline. Since there is only a single
is that of finding the optimal packing of the job’s VMs on
in 3. THE FLEXPACK BATCH SCHEDULING

MECHANISM

Due to the highly combinatorial nature of the FlexPack
problem, we develop the FlexPack heuristic to execute in
two phases: scheduling and packing. The first phase starts
by deciding on a schedule of job execution for the batch,
which minimizes the broker’s costs while satisfying the jobs’
physical and temporal demands. Given that schedule, the
second phase finds the packing with minimum cost for each
individual job within the batch. This multi-phase approach
decouples the scheduling and packing problems, allowing us
to use existing efficient algorithms to solve each separately.

3.1 Packing Size of a Job

Due to the data-intensive nature of the workloads, the
number of VM slots dedicated to a job does not always
match the number of slots it demands. If the job utilizes
a subset of slots over multiple servers, the remaining slots
on these servers may be unsuitable for other jobs due to their
limited available bandwidth, as shown in Figure 2. However,
since the broker must maintain these ghost slots to control
the available bandwidth capacity of the server, we are con-
sidered a consequence of the job’s demand. We define the
packing size of a job as an estimate of the number of slots
it occupies, whether they are utilized for computing or only
maintained for bandwidth.

Algorithm 1 Estimating a job’s packing size.
1: function EffectiveSize((D, C, B), Jk)
2: for Each server i do
3: Compute:
4: \( EC(i, k) = \max \{x \leq c_i\} \) \( s.t. \) \( x(n-x) t_k \leq b_i \)
5: Compute:
6: \( Cost(i, k) = (n - EC(i, k)) \frac{\sum_{1 \leq j \leq m} d_{i,j}}{m-1} \)
7: Sort the servers according to their costs
8: Pick the server j; the a’s server according to cost
9: \( k = \) maximum subset of \( J_i \) that fits in server j
10: Return \( n_i/k \ast c_j \)

The key to computing a job’s packing size is to estimate
the server capacities at the time of the packing, and accord-
ingly an estimate of the minimum number of servers that the
job can be packed into. The packing size of a job is com-
puted using a lighter version of the Greedy NCP heuristic
(G-NCP) [8], as presented in Algorithm ???. The available
servers are sorted in non-decreasing order according to their
accommodation costs for the job, and the first server is cho-
sen for packing the job’s VMs. By finding the largest subset
of the job’s VMs that can be placed into that minimum cost
server, we can estimate the maximum number of slots to be
required by the workload.

The cost of a server i for accommodating the VMs of a

physical and temporal demands. Given that schedule, the
second phase finds the packing with minimum cost for each
individual job within the batch. This multi-phase approach
decouples the scheduling and packing problems, allowing us
to use existing efficient algorithms to solve each separately.
job $k$ is defined as,

$$\text{Cost}(i, k) = (n - EC(i, k)) + \frac{\sum_{1 \leq j \leq m} d_{i,j}}{m - 1}$$

In the above, the effective capacity of a server $i$ according to job $k$’s requirements is represented by $EC(i, k)$, and $d_{i,j}$ represents the distance between servers $i$ and $j$. Moreover, the effective capacity of a server is defined as,

$$EC(i, k) = \max \{ x \leq c_i \}, \text{such that } x(n - x) t_{ik} \leq b_i$$

In the above, $t_k$ is job $k$’s average VM-pair bandwidth, and $c_i$ and $b_i$ are the computing and bandwidth resources available to the servers respectively.

Since the service broker capacities are not fixed, but depend on the time step considered and the jobs already packed in the servers during that time step, we define a knob-parameter $0 \leq \alpha \leq 1$ to denote the servers’ level of compactness. A larger value of $\alpha$ leads to an over-estimation of the job’s packing size, which facilitates the packing of the job, but might lead to an increase in the schedule’s length. Inversely, a smaller value of $\alpha$ improve scheduling, but might lead to packing failures.

### 3.2 Scheduling in FlexPack

By the definition of a job’s packing size, the dimensionality of the job’s demands is reduced into a size and span value. Thus, any existing approach for scheduling a batch of parallel jobs on multi-processors can be adopted to decide on the schedule with minimum makespan. In this section, we further exploit the reduced dimensionality of the job demands, and develop an efficient scheduling heuristic inspired from 2D strip packing problems. This developed scheduling heuristic has the added advantage of handling varying server capacities within a single time epoch.

In the 2D Strip Packing (2D-SP) problem, also referred to as the Orthogonal Stock-Cutting Problem, 2D rectangular items need to be packed into stock sheet with fixed width, but unbounded length/height [18, 6]. Each packed item must completely fit in the stock sheet, and cannot overlap with any other item. The objective of the 2D-SP problem is to minimize the maximum length of the stock sheet used. The problem is known to be NP-hard, and various heuristics [15, 6, 11, 37] and meta-heuristic [10, 27] approaches have been proposed to solve it. In our work, we use techniques from the Best Fit Strip Packing (BFSP) heuristic presented in [11], in which gaps within the stock sheet are filled with the best fitting rectangular item that minimizes wasted areas.

In the scheduling phase of FlexPack, the broker’s resources can be represented as a broker sheet, in which its width represents the VM slots it offers, and the unbounded length represents time. Moreover, given a job’s packing size and execution span, each job can be represented by a rectangle with its width representing the packing size, and its length representing its span. Now, the problem of scheduling the jobs with minimum schedule makespan is equivalent to that of the 2D strip packing problem to minimize the length of broker sheet used.

Due to the special nature of the scheduling problem, job rectangles can be cut into multiple horizontal strips during packing. Therefore, the lowest consistent gap is defined as the non-empty set of VM slots, not necessarily on the same server, with the earliest availability. Given the earliest gap with some capacity and duration, the best fitting job rectangle is picked from the set of feasible jobs to fill that gap. In Algorithm ??, the best fit can be determined using one of two metrics, the job’s area or the job’s packing size ratio. A job’s area is the product of its demanded size and execution span, and it’s best used in broker settings with very low collocation rates. On the other hand, a job’s packing ratio is the ratio between its packing size and its true size, and it’s almost always superior that the former metric. Finally, when the best fitting job is found, it might fit the gap perfectly, or might be smaller than the gap. The first case is straightforward, in which the job is perfectly placed in the gap, and in the second case, the job is placed at the top-left-most corner of the gap.

### 3.3 Resource Allocation in FlexPack

The second phase of the FlexPack heuristic is to use the G-NCP algorithm [8] to compute an efficient packing for each workload individually, at the time it is scheduled to start. To reduce packing failures, the workloads are sorted according to the ratio between their packing size and true size before packing, and jobs with higher ratio are packed first. Since a job with packing size closer to its real size indicates a lower bandwidth demand, it’s beneficial to delay its packing to minimize the overall packing cost.

If a job is successfully packed, the resources are reserved from the service broker, and the packing of the next job is attempted. If a job was not successfully packed, it is inserted into a new batch of jobs. After the attempted packing of all jobs is finalized, the two phases of the FlexPack heuristic are repeated with the jobs in new batches.

### 3.4 Payment Model in FlexPack

Since the extra costs incurred by the broker from ghost slots are not only caused by the workloads’ demands, but also the resource allocation decisions made by the broker, it cannot be simply divided over the workloads according to their demand. To guarantee fair pricing, and to provide incentives to workloads to share their true maximum temporal-flexibility, the extra cost has to be distributed over the workloads in the batch efficiently. Moreover, since the time-flexibility of a workload allows for better resource allocation, each workload should be rewarded for being flexible, by only paying for a portion of the ghost slots according to its contribution to that cost.

The key of minimizing the packing costs in our model is to exploit the reported flexibility of the workloads, i.e. whether they allow delayed deadlines or need to start right away. Increased workload flexibility allows for better packing, and
minimized broker cost, but also increases the workload’s inconvenience. Since a workload’s true deadline is private to its controlling agent, a rational agent might report less flexibility untruthfully, if it increases its own utility. Therefore, we propose using mechanism design principles, specifically the Clarke-Pivot rule used in VCG mechanisms [35], to give the agents an incentive to reveal their private information truthfully to the broker, i.e. the decision maker, and avoid strategic signaling.

In FlexPack, agents report their flexibility values upon arrival to the system, and cannot change them during the scheduling. Formally, we say that each agent $i$ has a privately known type $\theta_i$ that corresponds to that agent’s private flexibility value, and we denote by $\Theta_i$ the space of all of agent $i$’s possible types. An agent’s preference of a certain outcome is defined by its utility function, $u_i : \Theta_i \times O \rightarrow \mathbb{R}$, where $u_i(\theta_i, o)$ represents the agent’s utility for outcome $o \in O$ when the agent has type $\theta_i \in \Theta_i$. An agent’s utility function is publicly known by all other agents, but the exact utility of an agent for an outcome is only privately known by the agent as it depends on its true type.

We define an agent’s cost contribution as a measure of the effect of its workload on the quality of the packing, and thus on the broker’s cost. An agent’s cost contribution is computed using the Clarke-Pivot rule as the difference between the broker’s cost from a schedule with the agent’s workload, the broker’s cost from a schedule excluding the agent’s workload, and the base cost of the agent.

$$p_i(\theta_i, o) = \text{BrokerCost}(\Theta_i, o) - \text{BrokerCost}(\Theta_i \setminus \{i\}, o)$$

In which, $\theta_i \in \Theta_i$ is the reported value of flexibility by agent $i$.

We define an agent’s utility of an outcome $u_i : \Theta_i \times O \rightarrow \mathbb{R}$ as the difference between the payments computed for the agent, if it reports its true value of flexibility, and its reported value of flexibility.

$$u_i(\theta_i, o) = p_i(\theta_i, o) - p_i(\theta'_i, o)$$

**Theorem 2.** The payment model is incentive compatible, and individually rational.

**Proof.** Incentive Compatibility. An agent $i$ can increase his utility by reporting an untruthful flexibility value $\theta'_i$ to decrease its payment. As we had just shown in Theorem 3, an agent’s payment never increases if the agent reports a later deadline. Therefore, it is never the case that the agent’s utility would decrease if its maximum deadline is reported.

Individual Rationality. The agent’s utility is always greater than or equal to zero since he is never penalized for being flexible.

**Theorem 3.** An agent is never penalized for being flexible.

**Proof.** Assuming that we can find the optimal schedules that minimize the packing cost, and increased flexibility of an agent, i.e. a later workload deadline, will never increase the broker’s cost, given that no other workload changes its deadline. This is due to the fact that if the packing cost is less when the workload is scheduled earlier, it will still be

---

4. SCHEDULING WITH ONLINE ARRIVALS

Given an online arrival of data-intensive workloads, we are faced of a challenge of how to schedule them in a way that minimizes the broker’s cost, and how to compute a workload’s payment efficiently. In this section, we use backfilling techniques to create batches of jobs in an online setting.

4.1 Backfilling for Scheduling

In online settings, with jobs waiting in a queue to be served, backfilling algorithms [40, 39] are used to skip jobs at the head of the queue to schedule smaller jobs to increase the system’s utilization, with a guarantee that the jobs at the head of the queue never starve. There many variations of backfilling algorithms according to the lookahead allowed in the queue, the limitations on the scheduling of jobs at the back of the queue, and the non-starvation guarantees provided to jobs at the head of the queue. We propose creating batches from jobs waiting for service in the system queue, by optimally picking the set of jobs that would maximize the utilization of the broker’s resources with minimum communication cost.

In an online setting, jobs coming into the system may have different deadlines, but they are all assumed to be ready to start at their arrival. Jobs are placed in a queue according to their deadlines, to schedule jobs with earliest deadlines first minimizing their rejection rate. When a running job finishes its execution and exists the system at time step $t$, a scheduling decision is to be made, and the job at the head of the queue is retrieved. According to the job’s deadline, one of three cases could occur. In the first case, the job’s deadline is exceeded ($t + s_i > d_i$), so it is rejected and removed from the queue. In the second case, the job’s deadline can be met only if the job can be started right away ($t + s_i = d_i$). In this case, the job is removed from the queue and the scheduler attempts to pack it using the Greedy-NCP algorithm[8]. If the packing is successful, the resources are allocated to the job for the duration of its execution span.

Finally, in the third case when the deadline of the job is more relaxed ($t + s_i < d_i$), a better packing can be performed if the job is considered for resource allocation as part of a batch. To find the optimal batch of jobs, we define the queue lookahead value to be equal to the length of the queue, with the highest priority of resource allocation given to the job.
at the head of the queue. Then, we define an epoch to be \{t, t + 1, ..., d\}, and decide on the optimal resource allocation using the FlexPack problem. Since FlexPack is NP-hard, we approximate the solution using our proposed FlexPack heuristic, with consideration of the higher priority of the job at the head of the queue.

Upon choosing a batch of jobs, the jobs are rewarded for their flexibility of being scheduled as part of a batch, and the payment mechanism proposed above can be used to compute fair payments. Otherwise, i.e., the job is charged the exact cost incurred by the broker for its execution, including the charges for ghost slots.

5. PERFORMANCE EVALUATION

To evaluate the performance of our proposed mechanism in multiple settings, we developed a trace-based simulator that generates various problem instances, namely scenarios. In this section, we evaluate our scheduling algorithm in both the batch and online modes.

5.1 Setup

**Service Broker Model.** In our experiments, we assume that the service broker rents all of its VM slots from a single data center provider. The collocation factor of these VM slots, and the bandwidth available for them depends on the resource management models of the data center provider itself. To mimic the various collocation models of cloud providers, the service broker is defined by its average server capacity, i.e., average number of available VM slots per server, and its network load, i.e., the load of other applications in the data center on bandwidth availability. The server computing capacities are represented by a uniformly random variable based on the average server capacity, and the available network resources at each server is computed as a function of its computing capacity, and network load.

**Job Communication Model.** The size of a job, i.e. the number of VMs it requires, is a uniform random variable which is an appropriate distribution for evaluating the effect of the job size as adopted in [23]. The communication pattern of each job is adopted from the analysis of the application-level communication behavior of the different components of Bing.com, a large-scale Web application running in multiple data centers around the world, as presented in [9]. From these traces, we compile a set of 60 jobs, with communication and size properties as shown in Figure 3. Moreover, the temporal demands of a job, i.e., its execution span, arrival time, and deadline, are obtained from the NGS workload traces available on the Grid Workloads Archive [17].

**Performance Metrics.** We evaluate the efficiency of an allocation using several system-centric metrics, which represent the effect of the allocation on the service broker. The **broker cost**, represents the cost of the VM slots maintained by the broker to provide guaranteed execution predictability for the workloads. The **packing cost**, represents the cost of resource allocation as defined in [8]. The **packing rejection rate** represents the ratio between the number of jobs that failed in the resource allocation phase, and the total number of jobs in the batch. Moreover, for online experiments, we measure the **server utilization**, which represents the ratio between the number of used VM slots and the total number of VM slots originally offered by the service broker, and the **throughput**, which represents the average number of jobs served per unit time at steady state.

5.2 Batch Experiments with FlexPack

In all batch experiments, each batch contains 300 jobs, with resource and temporal demands synthesized from the traces mentioned above.

**Sensitivity Analysis.**

In the first set of experiments, we evaluate the efficiency of the scheduling mechanism in an offline setting, in which the demand of a single batch of workloads is predefined at the beginning of an epoch. The two phases of the FlexPack mechanism are executed, and jobs rejected in the second phase are rejected. The purpose of these experiments is to evaluate the sensitivity of the defined scheduling mechanism to various parameters. Each data point in the plots shown below represent the average of 30 experiments with the same initial settings.

**Effect of the best fit metric.** With a network load
of 5\(^5\), and 100 brokered VM slots. The results in Figure 4 represent the effect of the average server capacity on the broker cost, packing failure rate, and the packing cost, with the packing size of each job computed with \(\alpha = 0\). Lower average server capacity indicates a lower collocation rate at the data center provider, which leads to a slight over-estimation of the job’s packing size, even with \(\alpha = 0\). This over-estimation leads to inefficient packing, longer schedules, and increased broker’s costs. Although the packing ratio metric causes increased broker costs with lower collocation rates, it is superior in every other aspect. Using the packing ratio metric reduces the packing rejection rate by a factor of 9, and reduces the packing costs by 20%.

\[ \text{Effect of } \alpha \text{ in packing size.} \] In the second experiment, we evaluate the effect of \(\alpha\) on the computation of the packing size, and consequently the scheduling. The results in Figure 4 represent the effect of \(\alpha\) on the broker cost and packing failure rate, with varying average server capacity and network load. The variance in performance between the two extreme values of \(\alpha\) is exemplified with higher network load. That is the advantage of that knob parameter, as it allows for the packing of more jobs when needed.

\[ \text{Figure 4: Using the ratio between packing size and true size optimizes the packing even for } \alpha = 0. \]

(a) Lower server capacity leads to a slight over-estimation of packing size, which affects the broker costs.

(b) The packing ratio metric consistently leads to lower packing ratio.

(c) The packing ratio metric consistently leads to lower packing costs.

\[ \text{Figure 5: The effect of } \alpha \text{ is exemplified with varying network load.} \]

(a) Alpha has minimal effect with variation in server capacity. The sudden decrease in broker costs with \(\alpha = 0\) is caused by the high packing failure rate shown below.

(b) The sudden increase in the failure rate is caused by the extreme under-estimation of the packing size of the jobs.

(c) Effect of alpha starts to become clear with varying bandwidth availability.

(d) Higher values of \(\alpha\) lead to safer packing choices.

\[ ^5\text{The network load ranges between 5 and 20} \]
**Fairness of Pricing.**

To evaluate the fairness of the adopted pricing model, i.e. that it considers the contribution of the workload’s temporal-flexibility on the total broker cost, we compare it to the corresponding Shapley value. The Shapley value is the most commonly used approach for cost allocation and sharing, due to its theoretically proven efficiency and fairness [4]. However, due to its exponential nature of computation, it is not possible to compute the Shapley value in highly combinatorial optimizations in real-time. Therefore, we adopt the approach presented in [13] and later adopted in [30], to estimate the Shapley value within a 95% confidence interval in polynomial time.

In each scenario of this experiment, we generate a batch of 300 jobs, with resource and temporal demands obtained from the traces mentioned above. The broker offers a total of 100 VM slots, which are distributed over a set of servers. The average slot capacity of each server depends on the total number of servers in the scenario. The results shown in Figure 6 represent the payments computed for each job in a single scenario normalized by the value of the highest payment computed by the estimated Shapley approach. As expected, the payments computed from both approaches are not equal, but the distributions of payments are similar. Moreover, the VCG-based approach usually yields larger payments, hence higher broker profit.

![Figure 6](image)

Figure 6: Although VCG-based payments are a bit higher than their corresponding estimated Shapley value, the workload’s contribution is fairly considered in its value.

Moreover, to exemplify the fairness of the VCG-based payment model in practical situations, we compute the Pearson correlation coefficient between its computed payments and those computed by the estimated Shapley value, with varying server capacities. The increased correlation with decreased solution space exemplifies the practicality of our approach, since in practice the expected average server capacity, as computed by the broker, is assumed to be low [33].

### 5.3 Online Experiments

Figure 8 compares the performance of the FlexPack mechanism to the Earliest Deadline First (EDF) scheduling algorithm. For this online experiment, the service broker is set with 100 VM slots, an average server capacity of 10, and a network load of 5. The temporal properties of the jobs are generated from the NGS workload traces[17], including their arrival times and deadlines, and their communication properties are generated from the Bing.com traces [9]. In an online setting, jobs arriving to the system may have different deadlines, but they are all assumed to be ready to start at their arrival. Jobs are placed in a queue according to their deadlines, to schedule jobs with earliest deadlines first minimizing their rejection rate. When a running job finishes its execution and exists the system, the next job is picked
according to the scheduling algorithm adopted. The FlexPack online algorithm creates batches of jobs, and decides on the next best job to pack from the batch. The EDF algorithm simply picks the next job with the nearest deadline, and packs it using the G-NCP algorithm.

6. RELATED WORK IN CONTEXT

Resource Management in Distributed Systems.

Classically resource allocation in distributed systems was performed in an opportunistic manner upon the workload’s arrival to the system [20, 14]. The physical resources are allocated to a workload according to its on-time demand, which varies according to the temporal characteristics of the workload, and the availability of physical resources.

The first resource management system to propose advanced reservations of resources at the time of a workload scheduling was GARA [21], which was implemented as part of the Globus [20] infrastructure to guarantee the QoS of the workloads. In systems with advanced reservations, workloads specify their estimated demand through the period of their executions, usually associated with their temporal constraints. This demand specifications are used to allocate resources for the job for the duration of its execution. Later this advanced reservation approach was adopted in distributed systems to provide fairness and locality guarantees for workloads in Quincy [29], to provide performance predictability guarantees in SecondNet [26], and Oktopus [7], and to provide efficient utilization of resources and fairness in Tetris [25].

Similar to the model proposed in [26], we believe that performance predictability guarantees for data-intensive workloads cannot be achieved without explicit details about the communication model of the job, as well as its temporal demand. Our work complements that model, as we consider our adopted brokerage model for resource management, and the NCP problem as a model for resource allocation. Moreover, we assume the existence of a bandwidth rate limiting service within the system, which enables the system to enforce its resource reservations according to the workload demand.

Economical Models of Resource Management.

Resource pricing in distributed systems follows various microeconomic models, such as fixed-priced, auctions, iterative combinatorial exchanges, and fair cost allocation models. The most classically used approach of pricing is the fixed-priced model, as that used in Amazon web services [1], in which jobs know the exact prices of the resources before requesting them. However, with the emergence of federated systems and multi-site clusters, and with the increased demand on the same small set of resources, there has been a trend to allow jobs to define their own payments as in Amazon’s spot pricing [3]. This user-centric model of allowing jobs to name their own prices, creates a competitive environment, in which job owners need to be strategic and rational.

The most commonly used approach of allowing jobs to name their own prices is through auctions; single-bid, combinatorial, etc. In systems such as Bellagio [5], and Nimrod [12], jobs submit their bids on resources, and a centralized authority acts as the auctioneer, optimizing for the system’s utility. In systems such as in [36, 22], optimizations are done locally, and the global decisions are decided in a distributed approach using consensus-based algorithms. Moreover, mechanism design approaches [35] have also been used to incentivize jobs to report their true valuations of being scheduled and allocated resources as in [19, 32].

In the work by Ishakian et al. [31], a workload is represented by a work flow graph of its smaller tasks and its temporal demand flexibility. The authors optimize the allocation of resources according to the costs of operation, and propose a payment model based on the estimated Shapley value [13], in which each customer ends up paying a marginal cost of the resources utilized. Similar to their approach, in our work we optimally allocate resources to the workloads, and distribute the broker’s cost over the paying workloads according to their demand and temporal flexibility. However, we adopt a VCG-based approach with guaranteed incentive compatibility and individual rationality, which computes fair payments with much reduced computation complexity.

7. CONCLUSIONS

Considering the problem of scheduling and allocating resources to a batch of data-intensive workloads in a cloud brokerage environment with minimum broker cost, we define the FlexPack problem, and develop a heuristic approach to approximate it. By defining the packing size of a data-intensive workload, we enable the adoption of any efficient multi-processor parallel job scheduling algorithm. The defined VCG-based payment mechanism for our adopted cloud brokerage environment allows for fair distribution of the broker’s incurred costs over the participating workloads. Moreover, we confirmed via experimental evaluation that the fairness of our VCG-based payments is comparable to the corresponding estimated Shapley values [13], with the advantage of reduced computation complexity.

8. REFERENCES


D. o. s. Delft University of Technology, Faculty of EEMCS, P. Computer Technology, and D. Systems. Gwa-t-5 ngs.


D. o. s. Delft University of Technology, Faculty of EEMCS, P. Computer Technology, and D. Systems. Gwa-t-5 ngs.


