Big Data Aware Virtual Machine Placement in Cloud Data Centers

Logan Hall, Bryan Harris, Erica Tomes, and Nihat Altiparmak
Department of Computer Engineering and Computer Science
University of Louisville, KY 40292, USA
{logan.hall,bryan.harris.1,erica.tomes,nihat.altiparmak}@louisville.edu

ABSTRACT

While society continues to be transformed by big data, the increasing rate at which this data is gathered is making processing in private clusters obsolete. A vast amount of big data already resides in the cloud, and cloud infrastructures provide a scalable platform for both the computational and I/O needs of big data processing applications. Virtualization is used as a base technology in the cloud; however, existing virtual machine placement techniques do not consider data replication and I/O bottlenecks of the infrastructure, yielding sub-optimal data retrieval times. This paper targets efficient big data processing in the cloud and proposes novel virtual machine placement techniques, which minimize data retrieval time by considering data replication, storage performance, and network bandwidth. We first present an integer-programming based optimal virtual machine placement algorithm and then propose two low cost data- and energy-aware virtual machine placement heuristics. Our proposed heuristics are compared with optimal and existing algorithms through extensive evaluation. Experimental results provide strong indications for the superiority of our proposed solutions in both performance and energy, and clearly outline the importance of big data aware virtual machine placement for efficient processing of large datasets in the cloud.

KEYWORDS

virtualization; big data; cloud computing; storage systems

1 INTRODUCTION

Massive amounts of data are generated everyday by various sources including sensors, Internet transactions, social networks, Internet of Things (IoT) devices, video surveillance systems, and scientific applications. Many organizations and researchers store such data to enable breakthrough discoveries in science, engineering, and commerce. Today's most critical applications, including genome analysis, climate simulations, drug discovery, space observation & imaging, and numerical simulations in computational chemistry and high energy physics, are examples of data intensive applications dealing with large datasets commonly referred to as big data.

BDCAT'17, December 5–8, 2017, Austin, TX, USA

ACM ISBN 978-1-4503-5549-0/17/12…$15.00
https://doi.org/10.1145/3148055.3148057

Big data is generally stored in clusters of computers using distributed file systems [1, 2]. First, the dataset is divided into equal size disjoint chunks (~128 MB), chunks are replicated (~3 replicas), and distributed across nodes of the cluster to ensure scalability, availability, and reliability. Since the data to be processed is very large, an initial approach in efficient big data processing was to send the computation to the data and to retrieve data locally. However, existing high speed networking interconnects can provide transfer bandwidth higher than the storage throughput of even new generation NVMe devices, and can make the storage subsystem the cause of the bottleneck [3, 4]. The completion time of distributed big data processing applications is highly affected by the data retrieval times of individual nodes, and the data access bottleneck can lie both in storage and networking subsystems.

Cloud computing offers scalable big data storage and processing opportunities for academia and industry [5, 6]. Various scientific applications dealing with big data have already been deployed in the cloud recently [7]. For increased computer resource utilization, efficiency, and scalability, virtualization is used as a base technology by the cloud providers, and the data chunks of big data applications running in the cloud are retrieved and processed by virtual machines. An important scheduling decision in virtualized cloud data centers involves efficient mapping of the virtual machines (VM) having computer resource requirements to the physical machines (PM) with available resources. Various VM placement algorithms with different objectives were previously proposed, such as energy consumption minimization, network transfer minimization, economic cost minimization, performance maximization, and resource utilization maximization [8]. However, none of these techniques were designed considering applications processing big data, where data chunks are large, replicated, and both storage device and the network bandwidth can be the cause of the bottleneck in data retrieval. In order to perform efficient big data processing in the cloud, VM placement should be carefully performed by considering the data retrieval times of the virtual machines.

This paper deals with efficient big data processing in the cloud and proposes big data aware VM placement. Given a set of virtual machines with computer resource and data requirements, our aim is to determine the VM placement by minimizing the maximum data retrieval time of the VMs. Our proposed VM placement scheme considers data replication of the distributed file system, performance of the storage devices, and the network bandwidth.

2 PRELIMINARIES AND RELATED WORK

Big data aware virtual machine placement is closely related to two sub-problems: Replicated Data Retrieval Problem (RDRP) and Virtual Machine Placement Problem (VMPP).

2.1 Replicated Data Retrieval Problem (RDRP)

In a distributed system composed of N physical machines PM_1, $PM_2, \ldots, PM_N$, a virtual machine VM residing in one of the physical machines requests Q data chunks $D_1, D_2, \ldots, D_Q$ to be retrieved. Each data chunk is previously replicated r times and distributed across the physical machines using a distributed file system, such as Hadoop Distributed File System (HDFS) [2]. In the replicated data retrieval problem (RDRP), the aim is to decide which physical machine (replica) should serve each data chunk. The solution is called performance-optimal if the specified retrieval decision results in the minimum total retrieval time. In order to minimize the total retrieval time of all chunks, the maximum retrieval time for each physical machine used in retrieval should be minimized since the retrieval operation is performed in parallel. Performance-optimal RDRP can be expressed in linear form as follows:

$$\textbf{Minimize: } R$$
$$\textbf{Subject to: } \sum_{j \in P_i} B_{ij} = 1; \quad i = 1, \ldots, Q$$
$$R - R_j \geq 0; \quad j = 1, \ldots, N$$

B_{ij} is a binary variable which is set to 1 if chunk i is retrieved from physical machine PM_j, and set to 0 otherwise. Therefore, B_{ij} represents the final retrieval schedule (replica selection). The first constraint ($\sum_{j \in P_i} B_{ij} = 1$) ensures that every chunk i is retrieved from a single physical machine j, where P_i denotes the set of PMs holding a replica of chunk i. R_j in the second constraint represents the cost of retrieval from PM_j and it can be calculated as $R_j = S_j \cdot L_j$, where S_j denotes the single chunk retrieval cost from PM_j and $L_j = \sum_{i=1}^{Q} B_{ij}$ holds the number of chunks retrieved from PM_j. Finally, minimizing R guarantees that the maximum of R_j is minimized due to our second constraint ($R - R_j \geq 0$).

2.2 Virtual Machine Placement Problem (VMPP)

In virtualized cloud data centers, virtual machines (VM) with resource requirements such as CPU and memory are mapped to physical machines (PM) with available resources by satisfying single or multiple objectives. According to a recent survey [8], the most popular objective function aims to minimize the energy consumption of the data center, where 50% of the surveyed work focused on energy consumption minimization. A popular way to reduce energy consumption in virtualized cloud data centers is by powering down idle physical machines that are not holding any virtual machines. Therefore, we present the Virtual Machine Placement Problem (VMPP) with the objective of minimizing the number of physical machines used in the placement as follows.

Given a set of virtual machines $VM_1, VM_2, \ldots, VM_M$ with resource demands (CPU cores, memory, etc.) and a set of physical machines $PM_1, PM_2, \ldots, PM_N$ with resource capacities, VMPP aims to map VMs to PMs by satisfying the resource demands of the VMs, by respecting the resource constraints of the PMs, and by minimizing the number of PMs used in the mapping. By respecting the resource constraints, a single PM can hold multiple VMs. Based on this definition, VMPP can be formulated in linear form as:

$$\textbf{Minimize: } \sum_{j=1}^{N} I_j$$
$$\textbf{Subject to: } \sum_{j=1}^{N} X_{ij} = 1; \quad i = 1, \ldots, M$$
$$U_{jk} \leq C_{jk}; \quad j = 1, \ldots, N; \quad k = 1, \ldots, T$$

X_{ij} is a binary variable which is set to 1 if the VM_i is mapped to the physical machine PM_j, and set to 0 otherwise. Therefore, X_{ij} represents the final mapping. The first constraint ($\sum_{j=1}^{N} X_{ij} = 1$) ensures that every virtual machine is only mapped to a single physical machine. C_{jk} represents the capacity of PM_j for the resource type k, and U_{jk} represents the resource usage of PM_j for the resource type k. U_{jk} can simply be calculated as $U_{jk} = \sum_{i=1}^{M} (X_{ij} \cdot D_{ik})$, where D_{ik} indicates the resource demand of the VM_i for the resource type k. Therefore, our second constraint ($U_{jk} \leq C_{jk}$) guarantees that for each physical machine and resource type, resource usage does not exceed the resource capacity. Finally, our objective function makes sure that the number of physical machines used in the placement is minimized. This is guaranteed using a binary indicator variable I_j, which is set to 0 if $\sum_{i=1}^{M} X_{ij} = 0$, and set to 1 if $\sum_{i=1}^{M} X_{ij} > 0$. Therefore, I_j indicates whether PM_j is used in the placement or not, and $\sum_{j=1}^{N} I_j$ calculates the number of physical machines used in the placement. The mapping represented by the X_{ij} values is guaranteed to use the minimum number of physical machines satisfying the specified resource constraints.

Theorem 2.1. *VMPP is NP-hard.*

Proof. VMPP is equivalent to the d-dimensional Vector Bin Packing problem [9, 10], where VMs represent objects, PMs represent bins, and resources represent dimensions. Since d-dimensional VBM is NP-hard, so is VMPP. □

2.3 Related Work

In this section, we first provide the related work on replicated data retrieval and virtual machine placement problems, and then present the existing literature on data-aware virtual machine placement.

2.3.1 Replicated Data Retrieval. Replicated Data Retrieval Problem (RDRP) was first formulated in the work of Chen et al. [11] as a flow network optimization and solved in polynomial time using max-flow techniques [12]. This initial work assumed that the storage devices and the physical machine loads are homogeneous. Next, the problem was generalized to consider storage system heterogeneity (SSD/HDD), network delay, and physical machine loads, where a polynomial time max-flow solution was proposed for the generalized version [13]. This solution was further improved using parallelization and adaptive retrieval techniques [14–16].

In addition to the optimal solution, various heuristic based replica selection techniques were also proposed in literature and implemented in real settings without guaranteeing the optimal retrieval time. For example, *static* replica selection always retrieves chunks from a predefined replica [17]. HDFS [2] employs a *network-aware*

heuristic retrieving the chunks from the nearest replica based on the network topology. MongoDB [18] provides both options, where a static replica selection is employed by default, but an optional network-aware heuristic that uses the round-trip network delay is also provided. Finally, *load-aware* heuristics such as the *shortest-queue-first* algorithm [19] were commonly implemented in multimedia servers [20, 21]. These heuristics are generally used in homogeneous, centralized, or low-latency network settings where device queue lengths are compared and the device with the shortest queue length (fewest number of requests in its queue) is selected for data retrieval.

2.3.2 VM Placement. Virtualization is a proven resource sharing technology utilized in cloud data centers, and virtual machine placement is the heart of virtualization for the effective allocation of cloud resources. Therefore, various virtual machine placement techniques were proposed in recent literature [22–32]. Among these, energy efficiency is the most heavily investigated objective function for virtualized data centers [22–25]. In addition to energy consumption minimization, network traffic minimization [26–28], economical cost optimization [29], resource utilization maximization [30], performance maximization [31], and availability maximization [32] techniques were other popular objective functions studied for efficient virtual machine placement. Readers are directed to the literature review by Pires et al. [8] for an in-depth comparison and analysis of these virtual machine placement techniques.

2.3.3 Data-aware VM Placement. For efficient big data processing in the cloud, virtual machine placement should be carefully designed to consider the data retrieval times of the virtual machines. To the best of our knowledge, only a few works so far have dealt with data-aware virtual machine placement [33–36]. Among these, Piao et al. [33] and Zamanifar et el. [34] focused on minimizing data access latencies of the virtual machines and proposed heuristics that place them on the physical machines with better network bandwidth to the data. Alicherry et al. [35] also focused on data processing in the cloud and provided the optimal formulation for minimum data access using linear programming techniques; however, in order to simplify their formulation, they discarded resource requirements (CPU, memory, etc.) of the virtual machines and resource capacities of the physical machines. They reduced this simplified formulation to the linear assignment problem and solved it using the Hungarian algorithm [37]. In addition, they further added inter-VM distance constraints to their formulation, and provided heuristics for the resulting NP-hard problem. Kuo et al. [36] further improved their heuristic by providing a 2-approximation heuristic bounding the maximum access latency assuming that access latencies between the nodes satisfy the triangle inequality.

We note that none of these works consider data replication. They all assume that either a single replica exists for each data chunk or that replica selection is performed before or after virtual machine placement, affecting the optimality of the data transfer. In addition, Alicherry et al. [35] and Kuo et al. [36] simplify the problem further by discarding virtual machine resource requirements and physical machine resource capacities, which makes the entire virtual machine placement problem unrealistic. Finally, these works assume small data chunks and calculate data transfer cost by only considering the available network bandwidth or latency between the nodes, without considering the capabilities of the storage subsystem. Big data transfer cost is highly dependent on bottlenecks in the infrastructure, which can lie in the storage subsystem as well as in network bandwidth. Our proposed VM placement techniques consider VM resource requirements and PM resource capacities, data replication of the distributed file system, performance of the storage subsystem, and the available network bandwidth between the PMs.

3 BIG DATA AWARE VM PLACEMENT

The Replicated Data Retrieval Problem (RDRP) performs replica selection for only a single VM given the pre-placement of this VM on a specific PM, therefore it does not deal with the VM placement issue. On the other hand, Virtual Machine Placement Problem (VMPP) performs the VM placement without considering the data retrieval costs of the VMs and thus does not decide a retrieval schedule (replica selection). In order to perform efficient big data processing in the cloud, where virtual machines retrieve and process very large datasets, the VM placement should be aware of data retrieval costs. This paper proposes big data aware VM placement, which performs both VM placement and replica selection by considering data replica locations, storage retrieval performance, and network transfer performance to minimize data retrieval time.

3.1 Problem Formulation

We are given a set of virtual machines $VM_1, VM_2, \ldots, VM_M$ with resource demands (CPU cores, memory, etc.) and a set of physical machines $PM_1, PM_2, \ldots, PM_N$ with resource capacities. In addition, every virtual machine j requires a set of data chunk $D_1, D_2, \ldots, D_{Q_j}$ to be retrieved from the physical machines, where every chunk is replicated on r physical machines.

In the Big Data aware virtual machine Placement (BDP) problem, our aim is to place the virtual machines into the physical machines by minimizing the retrieval time of all data chunks of all virtual machines. The solution should also specify the retrieval schedule for each data chunk of every virtual machine by specifying the physical machine (replica) to be used, and it should also respect the resource constraints of physical machines since a single physical machine can contain multiple virtual machines.

In order to minimize the retrieval time of all chunks, we need to minimize the maximum retrieval time of the physical machine used in retrieval (as in RDRP) since the physical machines perform retrieval in parallel. Using the notation described in Table 1, BDP can be formulated as follows:

Minimize: R

Subject to:

$$\sum_{k \in P_{ij}} \sum_{l=1}^{N} B_{ijkl} = 1; \quad i = 1, \ldots, Q_j; \quad j = 1, \ldots, M$$

$$\sum_{i=1}^{Q_j} \sum_{k \in P_{ij}} B_{ijkl} = Q_j \cdot I_{jl}; \quad j = 1, \ldots, M; \quad l = 1, \ldots, N$$

$$U_{lt} \leq C_{lt}; \quad l = 1, \ldots, N; \quad t = 1, \ldots, T$$

$$R - R_k \geq 0; \quad k = 1, \ldots, N$$

Table 1: Notation

Not.	Description
M	Number of virtual machines: $VM_1, VM_2, \ldots, VM_M$
N	Number of physical machines: $PM_1, PM_2, \ldots, PM_N$
T	Number of resource types
B_{ijkl}	1 if chunk i of VM_j is transfered from PM_k to PM_l, 0 otherwise
Q_j	Number of chunks required by VM_j: $\{D_1, D_2, \ldots, D_{Q_j}\}$
Q	Total data requirement (in chunks) for all VMs; $Q = \sum_{j=1}^{M} Q_M$
r	Replication factor for the chunks
P_{ij}	Set of PMs holding a replica for chunk i of VM_j; $\lvert P_{ij} \rvert = r$
I_{jl}	1 when VM_j is placed in PM_l, 0 otherwise; $\sum_{i=1}^{Q_j} \sum_{k \in P_{ij}} B_{ijkl} = Q_j \cdot I_{jl}$
D_{jt}	Resource demand of the virtual machine j for the resource type t
C_{lt}	Capacity of the physical machine l for the resource type t
U_{lt}	Usage of the physical machine l for the resource type t; $U_{lt} = \sum_{j=1}^{M} (D_{jt} \cdot I_{jl})$
S_{kl}	Single chunk transfer time from PM_k to PM_l
L_{kl}	Number of chunks (load) transfered from PM_k to PM_l; $L_{kl} = \sum_{j=1}^{M} \sum_{i=1}^{Q_j} B_{ijkl}$
R_k	Time to transfer all chunks from PM_k; $R_k = \sum_{l=1}^{N} S_{kl} \cdot L_{kl}$
R	Optimal retrieval time of all data chunks of all VMs

B_{ijkl} is a binary variable which is set to 1 if chunk i of VM_j is transfered from PM_k to PM_l, and set to 0 otherwise. Therefore, B_{ijkl} represents the final retrieval schedule (replica selection). The first constraint ($\sum_{k \in P_{ij}} \sum_{l=1}^{N} B_{ijkl} = 1$) ensures that every chunk i of every VM_j is only transfered from a single PM_k to a single PM_l, where P_{ij} denotes the set of PMs holding a replica for chunk i of VM_j. The second constraint ($\sum_{i=1}^{Q_j} \sum_{k \in P_{ij}} B_{ijkl} = Q_j \cdot I_{jl}$) makes sure that all the chunks of a VM_j are transferred to a single PM_l, where I_{jl} is a binary variable which is set to 1 if VM_j is placed on PM_l, and set to 0 otherwise. As in VMPP, C_{lt} represents the capacity of PM_l for resource type t, and U_{lt} represents the resource usage of PM_l for resource type t. U_{lt} can simply be calculated as $U_{lt} = \sum_{j=1}^{M} (D_{jt} \cdot I_{jl})$, where D_{jt} indicates the resource demand of VM_j for resource type t. Therefore, our third constraint ($U_{lt} \leq C_{lt}$) guarantees that for each physical machine and resource type, resource usage does not exceed the resource capacity. R_k in the last constraint holds the time to transfer all requested chunks from PM_k and it can be calculated as $R_k = \sum_{l=1}^{N} S_{kl} \cdot L_{kl}$, where S_{kl} denotes the single chunk transfer time from PM_k to PM_l and $L_{kl} = \sum_{j=1}^{M} \sum_{i=1}^{Q_j} B_{ijkl}$ denotes the total number of chunks (load) retrieved from PM_k to PM_l. Finally, minimizing R guarantees that the maximum of all R_k values are minimized due to our last constraint ($R - R_{kl} \geq 0$).

This formulation uses NQr B_{ijkl} variables, NM I_{jl} variables, and an R variable. All variables except R are binary and the total number of unique variables is $NQr + NM + 1$. In addition, it uses a total of $Q + N(M + T + 1)$ constraints. This is a mixed integer programming formulation, which is classified as NP-hard [38].

4 LOW-COST HEURISTICS FOR BDP

In addition to the optimal solution, we also propose low-cost heuristics *bdp* and *ff-data*, which do not guarantee the optimality of the result, but are expected to achieve a solution close to optimal by performing greedy replica selection and greedy VM placement.

Algorithm 1 Best-Data VM Placement (***bdp***)

```
In: N, M, T, Q[ ], P[ ][ ], S[ ][ ], availability[ ][ ], demand[ ][ ], load[ ]
Out: placement[ ], retrieval[ ][ ], R
1:  retrieval_time ← 0
2:  sorted_vms ← VMs sorted (asc.) by number of chunks required
3:  for all vm in sorted_vms do
4:      best_cost ← ∞
5:      best_pm ← nil
6:      for pm ← 1 to N do
7:          if IsCompatible(vm, pm, availability, demand, T)
8:              max_cost ← retrieval_time
9:              for k ← 1 to N do
10:                 this_load[k] ← load[k]
11:             for c ← 1 to Q[vm] do
12:                 selected_pm ← GR(P[c][vm], pm, S, this_load)
13:                 this_load[selected_pm] += S[selected_pm][pm]
14:                 this_retrieval[c] ← selected_pm
15:                 if this_load[selected_pm] > max_cost
16:                     max_cost ← this_load[selected_pm]
17:             if max_cost < best_cost
18:                 best_cost ← max_cost
19:                 best_pm ← pm
20:                 for k ← 1 to N do
21:                     best_load[k] ← this_load[k]
22:                 for c ← 1 to Q[vm] do
23:                     best_retrieval[c] ← this_retrieval[c]
24:     if best_cost > retrieval_time
25:         retrieval_time ← best_cost
26:     placement[vm] ← best_pm
27:     for resource ← 1 to T do
28:         availability[best_pm][resource] −= demand[vm][resource]
29:     for c ← 1 to Q[vm] do
30:         retrieval[vm][c] ← best_retrieval[c]
31:     for k ← 1 to N do
32:         load[k] ← best_load[k]
```

Function 1 IsCompatible()

```
In: vm, pm, availability[ ][ ], demand[ ][ ], T
1: for resource ← 1 to T do
2:     if availability[pm][resource] < demand[vm][resource]
3:         return false
4: return true
```

4.1 Best-Data VM Placement (*bdp*)

The proposed Best-Data VM Placement (*bdp*) heuristic is shown in Algorithm 1, which aims to place VMs on the PMs in a greedy fashion depending on which PM yields the best retrieval time considering the previously placed VMs and their requests, as well as network and storage bottlenecks. Algorithm 1 first sorts the VMs (line 2) in ascending order of the number of chunks required by the VM. Ascending order allows the heuristic to balance the load of data retrieval across the PMs. Otherwise, if a VM with a large number of chunks were to be placed first, then a VM with a smaller

number of chunks would have more opportunity to be placed in an unbalanced way in terms of storage and networking performance. We also observed this behavior experimentally. The motivation behind this choice is to achieve a better retrieval performance; however, as every other VM placement technique, *bdp* might not be able to place all VMs due to resource contention, and sorting by the number of chunks does not help it much. If *bdp* cannot place all VMs, then the algorithm can be executed one more time by sorting by VM resource requirements in decreasing order. Such sorting is expected to achieve a tighter fitness (as performed in Algorithm 2); however, it is also expected to increase the chance of uneven load and network/storage bottlenecks on tightly fitted PMs.

For every VM in line 3, the heuristic iterates through every PM in line 6 and checks its compatibility based on VM resource requirements in line 7 through Function 1. If the PM is compatible, it hypothetically places the VM on that PM in lines 8–16, and also selects replicas using a greedy retrieval technique, as shown in Function 2. The idea is to consider a data retrieval cost (R_k) as in the LP formulation, but to update the PM loads (L_{kl}) in a greedy manner. Lines 17–23 maintain the minimum maximum retrieval time and its associated VM placement and replica selection. Lines 24–25 update the retrieval time of the entire data set. The hypothetical placement that yields the minimum data retrieval cost is then selected for placement (line 26) and PM resource availability is updated (lines 27–28). The associated replica selection is used in lines 29–30. After each placement, the loads of the PMs are updated (lines 31–32) so that they can influence the next VM placement. The worst-case time complexity of *bdp* is $O(N^2MT + NTQr + MlogM)$.

4.2 First Fit Data (*ff-data*)

Algorithm 2 presents the First Fit Data (*ff-data*) heuristic. The motivation behind proposing *ff-data* is to achieve a better fitness in VM placement that reduces the total number of PMs used, thus yielding a reduced energy consumption. In addition, our aim is to propose an alternative heuristic to *bdp* and evaluate their performance in both energy consumption and data retrieval.

As with *bdp*, *ff-data* also starts by sorting VMs in line 1; however, the sorting is performed here in decreasing order by resource requirements of the VMs. In order to convert multi-dimensional resource requirements of the VMs to a scalar, it uses the *FFDAvgSum* technique proposed by Panigrahy et al. [39], which calculates the VM weights using $w(j) = \sum_{t=1}^{T} a_t D_{jt}$ for a VM_j where $a_t = \frac{1}{M}\sum_{j=1}^{M} D_{jt}$ represents the average demand for resource type t. The sorting is performed in decreasing order so that the VMs with the largest resource requirements are placed first, as there may be a limited number of compatible PMs. In bin packing theory, FFD-based algorithms are known to be effective in practice, and guarantee to find an allocation with at most $\frac{11}{9}OPT + 1$ bins in the one dimensional case [40].

For every VM in line 2, the first compatible PM (always starting with PM ID 1) is determined in lines 3–4. The placement is performed in line 5 and the resources are updated in lines 6-7. Once the placement is performed for a particular VM, replicas are selected in lines 9–11 using the greedy retrieval technique (Function 2). Finally, line 12 updates the PM loads so that they can influence the next VM placement. The worst-case time complexity of *ff-data* is $O(NMT + Qr + MlogM)$. Clearly, this is an improvement over *bdp*'s time complexity of $O(N^2MT + NTQr + MlogM)$ with a possible loss in data retrieval performance.

Algorithm 2 First Fit Data (***ff-data***)

In: N, M, T, Q[ ], P[ ][ ], S[ ][ ], availability[ ][ ], demand[ ][ ], load[ ]
Out: *placement*[ ], *retrieval*[ ][ ]

```
1:  sorted_vms ← VMs sorted (desc.) by resource requirements
2:  for all vm in sorted_vms do
3:      for pm ← 1 to N do
4:          if IsCompatible(vm, pm, availability, demand, T)
5:              placement[vm] ← pm
6:              for resource ← 1 to T do
7:                  availability[pm][resource] −= demand[vm][resource]
8:              break
9:      for c ← 1 to Q[vm] do
10:         selected_pm ← GR(P[c][vm], placement[vm], S, load)
11:         retrieval[vm][c] ← selected_pm
12:         load[selected_pm] += S[selected_pm][host_pm]
```

Function 2 GR() – Greedy Retrieval

In: replica_pms, host_pm, S[ ][ ], load[ ]
Out: *selected_pm*

```
1:  min_cost ← ∞
2:  selected_pm ← nil
3:  for all candidate_pm in replica_pms do
4:      cost ← load[candidate_pm] + S[candidate_pm][host_pm]
5:      if cost < min_cost
6:          min_cost ← cost
7:          selected_pm ← candidate_pm
8:  return selected_pm
```

5 BOTTLENECK ANALYSIS FOR RETRIEVAL

A critical component of our proposed algorithms is being able to estimate the S_{kl} values representing the single chunk transfer time from PM_k to PM_l. Assuming a well engineered network and large data chunks (64 MB to 256 MB), as commonly used in big data analysis platforms, S_{kl} values are expected to be governed by the bottleneck of two important properties of the distributed system: (i) local storage system throughput for PM_k, and (ii) network bandwidth between PM_k and PM_l. Existing techniques generally disregard the storage system capabilities and only focus on the network, which is susceptible to yield incorrect results in big data analysis platforms especially with today's high speed networking interconnects and heterogeneous storage architectures composed of flash and spinning disks. In order to validate the importance of bottleneck analysis in the estimation of S_{kl} values, we performed a set of experiments with various storage and networking configurations. Our experiments ran on Linux physical machines (Ubuntu 16.04 LTS, kernel version 4.4.0) with four different storage architectures:

- Single HDD (Toshiba MG03ACA100 1 TB SATA 3)
- Single SSD (Intel S3510 800 GB SATA 3)
- 4 HDDs (4 × Toshiba MG03ACA100 1 TB SATA 3) as an `mdadm` [41] software RAID-10 array
- 4 SSDs (4 × Intel S3510 800 GB SATA 3) as an `mdadm` software RAID-10 array

Using these storage configurations, we calculated the transfer times of 1, 2, 4, and 8 chunks of 128 MB each, in three different scenarios: (i) local read, (ii) remote read via 1Gbps network, and (iii) remote read via 100Mbps network. For the local read experiments, chunks were read from the storage system using `dd` [42] once the page cache, directory entries, and the inodes were cleared from the memory using `sync && echo 3 > /proc/sys/vm/drop_caches`. For the remote read experiments, `nc` (`netcat`) [43] was utilized first to set up a TCP connection and next `dd` was used to pipe data to `nc` after clearing the memory cache as shown above. Table 2 displays the average results over 10 runs.

Table 2: Local/Remote Chunk (128 MB) Retrieval Times

Storage System	Local Read Times			
	1 chunk	2 chunks	4 chunks	8 chunks
4 SSDs, RAID-10	155ms	287ms	566ms	1.13s
4 HDDs, RAID-10	431ms	848ms	1.68s	3.27s
1 SSD	394ms	779ms	1.54s	3.09s
1 HDD	825ms	1.62s	3.24s	6.41s
Storage System	**Remote Read Times via 1 Gbps Network**			
	1 chunk	2 chunks	4 chunks	8 chunks
4 SSDs, RAID-10	1.37s	2.61s	5.20s	10.2s
4 HDDs, RAID-10	1.35s	2.63s	5.42s	10.2s
1 SSD	1.37s	2.60s	5.15s	10.2s
1 HDD	1.35s	2.61s	5.16s	10.2s
Storage System	**Remote Read Times via 100 Mbps Network**			
	1 chunk	2 chunks	4 chunks	8 chunks
4 SSDs, RAID-10	11.5s	22.9s	46.1s	91.3s
4 HDDs, RAID-10	11.5s	22.9s	46.1s	91.3s
1 SSD	11.5s	22.9s	46.1s	91.3s
1 HDD	11.5s	22.9s	46.1s	91.3s

As expected, retrieval times are directly proportional with the number of chunks to be transfered in all scenarios, for both SSD and HDD based storage systems since the chunk sizes are large (128 MB). In local reads, the SSD array achieves the fastest retrieval time by providing around 1 GB/s throughput, and the single HDD achieves the slowest retrieval time by providing around 165 MB/s. However, in both remote read scenarios, network transfer becomes the bottleneck since the 1 Gbps network can only provide 128 MB/s transfer rate and the 100 Mbps network can only provide 12.8 MB/s transfer rate. If the network bandwidth is the bottleneck, then the storage throughput does not affect transfer time as it can be seen from these remote read experiments since all four storage configurations provide a faster throughput. However, since high speed networking interconnects providing 10/40/100 Gbps are common in today's clusters, even SSD arrays that provide a few GB/s transfer rate can end up being the cause of the bottleneck.

In summary, these experiments emphasize the importance of bottleneck analysis in big data transfer, where both storage throughput and network bandwidth play an important role. We also measured the effect of switch delay and found that an extra switch adds ~12 ms per 128 MB chunk, which seems to be insignificant compared to the total transfer time of large chunks. We use these experimental results in our evaluation to estimate the S_{kl} values representing the single chunk transfer time from PM_k to PM_l.

6 EVALUATION

In this section, we evaluate the performance of the proposed heuristics by comparing their data retrieval and energy-efficiency with the existing techniques and the optimal retrieval values.

6.1 Experimental Setup

We performed simulations supported by real world transfer time calculations as discussed in Section 5. An in-house simulator, *vm-sim*, was designed to aid in this endeavor. Realistic workloads vary widely across organizations, making it difficult to generalize from any given set of real workloads. Following the works of Alicherry et al. [35] and Kuo et al. [36], we ran our algorithms on synthetic instances with a variety of different configurations to examine the behavior of our heuristics. Except the cases where heuristics are compared with the optimal performance values obtained using IBM's CPLEX LP solver [44], we repeated each of our experiments 100 times and averaged the results. Due to space limitations, we only share our experimental results for a selected subset of VM (M) and PM (N) values ($M = N = 32, 128,$ and 512), and the remaining cases were observed to be consistent with the presented arguments.

6.1.1 Network and Storage Configuration. In our evaluation, we used a similar set of storage and network configurations presented in Table 2 and discussed in Section 5. In order to observe situations in which storage is always the bottleneck, we replaced the 100 Mbps network configuration with a 10 Gbps network configuration since 1 Gbps and 100 Mbps cases were both bottlenecked at the network, generating similar scenarios. In addition, we eliminated the 4-HDD setup since it provided a comparable performance with the 1-SSD case. Finally, we also introduced heterogeneous storage and network configurations to further challenge the evaluated algorithms. As a result, we used three different network configurations: (i) 1 Gbps homogeneous, (ii) 10 Gbps homogeneous, and (iii) 1/10 Gbps heterogeneous (mixed). In homogeneous networks, all links have the same transfer rate, but in heterogeneous networks, the link rates are randomly selected between 1 Gbps and 10 Gbps. In addition, we used four storage configurations: (i) 1-HDD homogeneous, (ii) 1-SSD homogeneous, (iii) 4-SSDs homogeneous, and (iv) heterogeneous (mixed). In the homogeneous storage scenarios, all PMs have the same storage system; however in the heterogeneous scenario, storage systems of the PMs are randomly selected from the 1-HDD, 1-SSD, and 4-SSDs cases.

The bottleneck experiments performed in Section 5 allowed us to use real data transfer times in our experiments. In our simulator, we recorded the S_{kl} values between every PM_k to every PM_l in a matrix, which can easily be used and updated periodically in real settings based on observed transfer rates and the exponential averaging techniques. As in Section 5, we assumed a chunk size of 128 MB and simulated different network topologies by adding a random number of switches between nodes (1 to 5 switches) using the observed switch delay of 12 ms. For the 10 Gbps network configuration, we determined the transfer time between two PMs based on the storage throughput of the source PM and the additional switch delay observed during the data transfer, since all our storage configurations are slower than the 10 Gbps network. We ran our experiments on the cross product of our network and storage configurations. For comparison with the optimal data retrieval values, we ran our LP formulation for a limited subset of our experiments ($M = N =$ 16 and 32, 10 Gbps network) due to its long execution time.

6.1.2 Data Placement, Replication, and VM Data Requirements. Similar to relational databases, we used a two-dimensional grid to

represent the data placement scheme. Each cell in the grid represented a 128 MB chunk; and the value in a cell denoted the physical machine a chunk is stored in. The number of rows and columns in the grid was set to be equal to the number of physical machines. We used a replication factor of three in our experiments, and therefore needed three separate grids to represent every replica for each chunk. Data placement was accomplished using a periodic data allocation technique [45] similar to RAID-0 striping in two dimensions. Once the first replica of a chunk is placed on the physical machine, the second and third replicas are placed by shifting the location of the first replica, where a shift value of one is chosen in our experiments. For example, if chunk 0 is placed on physical machine 0, chunks 1 and 2 are placed on machines 1 and 2, respectively. We also experimented with random allocation in which replicas were randomly allocated to physical machines. Since the two allocation schemes yielded similar behaviors across experiments, we only share our experimental results for the periodic allocation and shifted periodic replication scheme due to space limitations.

Representing data placement as a grid was useful for determining virtual machine data requirements based on commonly used relational database query types. We used range queries due to their popularity. The size of the grid and the number of chunks required by each VM were varied based on the number of physical machines used in the system. The expected number of chunks to be processed by the VMs were set to $\frac{3N}{2}$, where N is equal to the total number of physical machines used in the experiment [13].

6.1.3 VM Resource Requirements and PM Capacities. Physical machines possess a limited number of computer resources, and virtual machines require a portion of these resources. We used two resource types in our evaluation, CPU cores and memory, and used the following Amazon EC2 instances [46] to determine our VM resource requirements: (i) *t2.small* (1 CPU Core, 2 GB Memory), (ii) *t2.medium* (2 CPU Cores, 4 GB Memory), (iii) *t2.large* (2 CPU Cores, 8 GB Memory), and (iv) *t2.xlarge* (4 CPU Cores, 16 GB Memory). Virtual machines were randomly selected from this pool of instances. Since physical machine resource capacities vary depending on the virtual machines that they host at any snapshot of the system, we randomly selected the PM capacities from 1 to 8 cores and 1 to 32 GB of memory. This random selection in every run and averaging the results over 100 runs allowed us to simulate physical machines with various pre-existing loads.

6.1.4 Algorithms. We implemented the following algorithms:

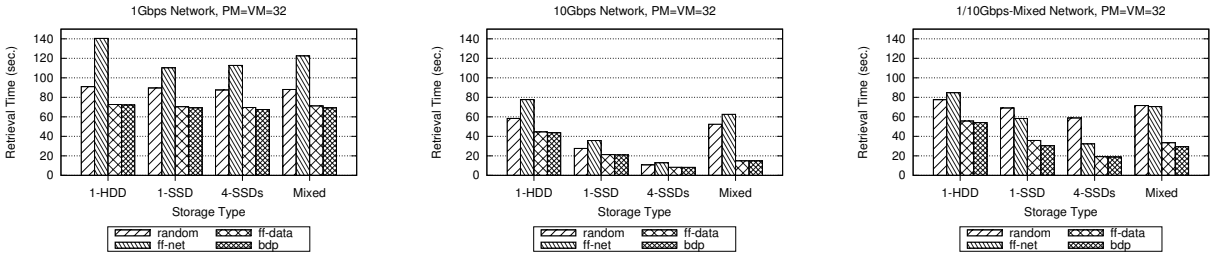
- ***random*** places VMs on randomly selected PMs. Local replicas are selected if available; otherwise, replicas are also selected randomly. The worst-case time complexity is $O(NMT)$.
- ***ff-net*** uses a first-fit decreasing strategy to place VMs on PMs [39], and it follows an HDFS-like network-aware replica selection strategy [2], where if a local replica exists, the data is retrieved locally; otherwise, it selects a replica from the physical machine with the smallest network transfer time to the host machine. If a tie occurs for the nearest replica, then the tie is broken randomly. The worst-case time complexity of *ff-net* is $O(NMT + Qr + MlogM)$.
- ***ff-data*** also uses a first-fit decreasing strategy to place VMs on PMs as shown in Algorithm 2; however, it uses a greedy replica selection that considers the retrieval cost of selecting the replica from each source PM as in Function 2. The source chosen is the one with the lowest retrieval cost considering the machine load and transfer time. The worst-case time complexity of *ff-data* is $O(NMT + Qr + MlogM)$.
- ***bdp*** uses a greedy strategy for placing VMs on PMs as shown in Algorithm 1. All PMs that satisfy VM requirements are considered for placement. Greedy replica selection is performed for each PM candidate and the PM placement that leads to the minimum total data retrieval time out of all PMs is chosen. The worst-case time complexity of *bdp* is $O(N^2MT + NTQr + MlogM)$.
- ***optimal*** implements the LP formulation from Section 3.1 using IBM CPLEX [44]. CPLEX's MIP engine uses the branch and bound technique, which is classified as NP-hard [38].

6.2 Experimental Results

6.2.1 Data Retrieval Performance. Figures 1, 2, and 3 present the data retrieval performance of the algorithms for various homogeneous and heterogeneous network and storage configurations for 32, 128, and 512 machines, respectively, where the x-axis represents the storage type and the y-axis represents the data retrieval time in seconds. In the homogeneous 1 Gbps network configuration case shown in Figures 1(a), 2(a), and 3(a), since network bandwidth (being slower than the throughput of all storage configurations) is the cause of the bottleneck, *ff-net* yields the worst performance and requires over 140 seconds more than the *random* algorithm to retrieve the same dataset for 512 machines. The reason for this lies in the tight fitness of the VMs over the PMs and poor replica selection performed by the *ff-net* algorithm, which prefers nearest replicas and generates bottlenecks in the PMs holding these replicas. *random* consistently performs better than *ff-net* since it yields a more uniform distribution over the PMs for both VM placement and replica selection. Both *bdp* and *ff-data* consistently perform better than the others since they balance the load on the PMs better. We also observed that *bdp* performs better than *ff-data* by up to 9 seconds.

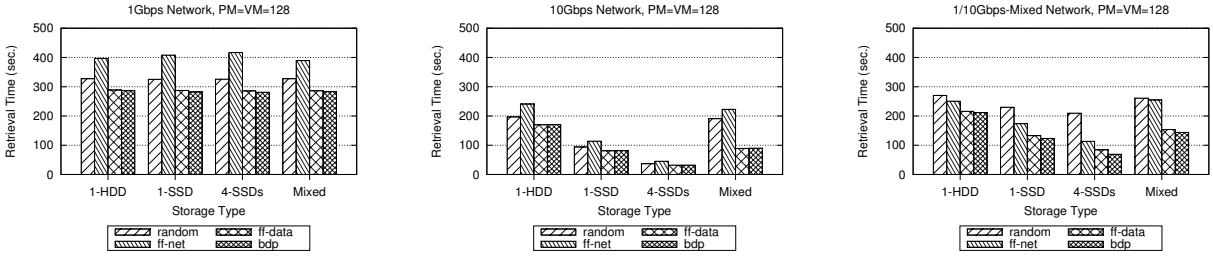
For the 10 Gbps homogeneous network configuration case as shown in Figures 1(b), 2(b), and 3(b), network is not the cause of the bottleneck but all storage systems generate lower data throughput than the available network bandwidth and become the cause of the bottleneck in data transfers. Therefore, the gap between *random* and *ff-net* narrows in this case; nevertheless, *random* still performs better due to the same reason as in the 1 Gbps case. It is possible to observe the storage bottleneck that *ff-net* experiences by paying attention to the performance improvement of *ff-net* as the storage system gets faster, especially for the 4-SSD case. For the faster storage system, the performance gap between *random* and *ff-net* is the smallest. The proposed *ff-data* and *bdp* heuristics again outperform the others since they are aware of storage bottlenecks in this case and they are able to retrieve replicas accordingly.

For the 1/10 Gbps heterogeneous network, shown in Figures 1(c), 2(c), and 3(c), *ff-net* passes *random* in performance, especially when the storage is faster since *ff-net* is network-aware and able to select better network links in retrieval compared to *random*. The proposed *ff-data* and *bdp* heuristics still outperform both *random* and *ff-net*, and the performance difference between *bdp* and *ff-data* becomes even larger (up to 36 sec.) in this heterogeneous case.



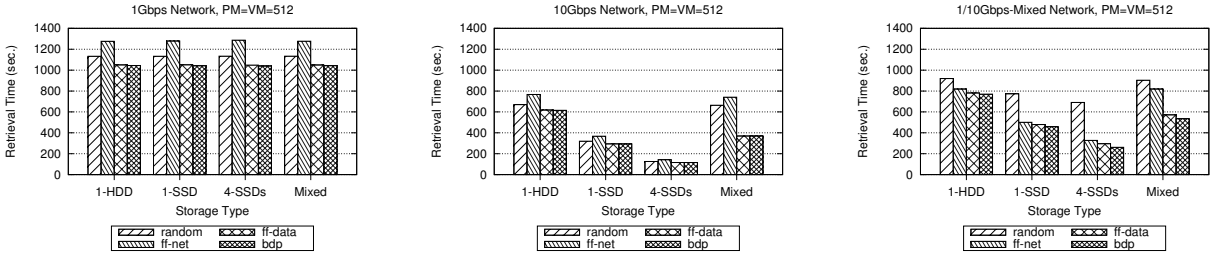
(a) 1 Gbps Homogeneous Network (b) 10 Gbps Homogeneous Network (c) 1/10 Gbps Heterogeneous Network

Figure 1: Data Retrieval Performance, M = N = 32



(a) 1 Gbps Homogeneous Network (b) 10 Gbps Homogeneous Network (c) 1/10 Gbps Heterogeneous Network

Figure 2: Data Retrieval Performance, M = N = 128



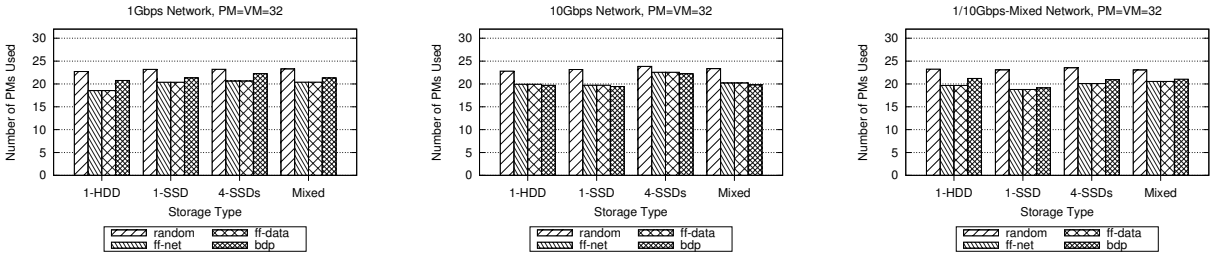
(a) 1 Gbps Homogeneous Network (b) 10 Gbps Homogeneous Network (c) 1/10 Gbps Heterogeneous Network

Figure 3: Data Retrieval Performance, M = N = 512

These results show that performing the VM placement together with a storage and network-aware replica selection technique clearly yields better data retrieval times, reaching up to 371 seconds better than the existing *ff-net* heuristic, and 429 seconds better than the *random* heuristic. This performance improvement is mainly due to accurate bottleneck estimation and load balancing of the proposed heuristics on the PMs.

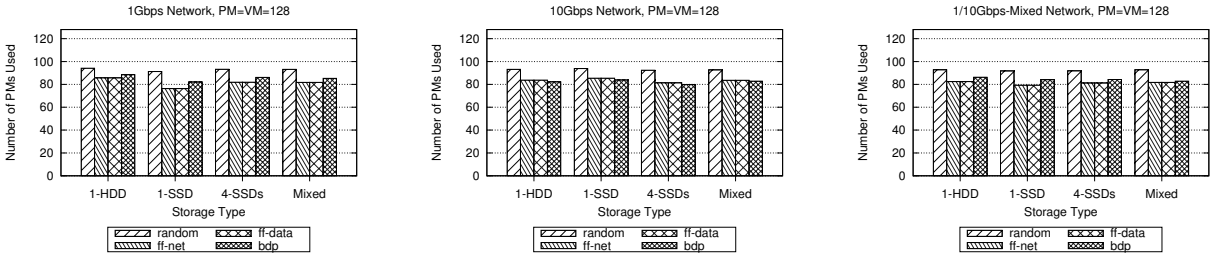
6.2.2 Energy Efficiency. We also evaluate the energy efficiency of the proposed algorithms by comparing the number of PMs used in the placement in Figures 4, 5, and 6 for 32, 128, and 512 machines, respectively, where the x-axis represents the storage type again but the y-axis represents the number of PMs used in the placement in this case. In terms of energy efficiency, *random* achieves the worst performance by using the most number of PMs in the placement in all cases. This is not surprising since *random* is expected to achieve a uniform distribution of VMs over PMs. First-fit based VM placement heuristics *ff-net* and *ff-data* both achieve the same energy efficiency as expected since they use the same VM placement strategy, and their energy-efficiency performance is slightly better than *bdp* for the 1 Gbps homogeneous network and 1/10 Gbps heterogeneous network cases as shown in Figures 4(a), 5(a), 6(a), and Figures 4(c), 5(c), 6(c), respectively. However, *bdp* achieves the best energy efficiency in the 10 Gbps homogeneous network case, as shown in Figures 4(b), 5(b), and 6(b), where the storage system is the cause of the bottleneck. This is mainly due to the fact that *bdp* places VMs over PMs that are closest to each other and therefore achieves a tight fit.

As also discussed by Ananthanarayanan et al. [3], with the availability of 40 and 100 Gbps network bandwidths in today's clusters, the storage system becomes the main source of the bottleneck in data transfers and even new generation NVMe solutions providing a few GB/sec storage throughput [4] cannot keep up with the available network bandwidth. Therefore, being aware of the storage subsystem bottlenecks and performing VM placement accordingly becomes crucial for efficient big data processing in private



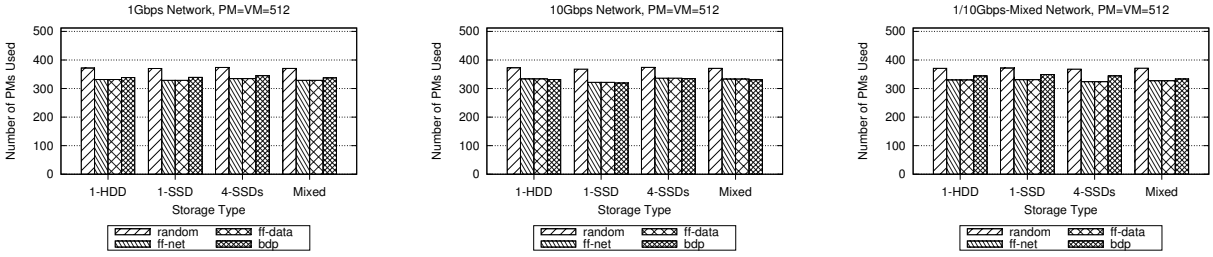
(a) 1 Gbps Homogeneous Network (b) 10 Gbps Homogeneous Network (c) 1/10 Gbps Heterogeneous Network

Figure 4: Energy Efficiency Performance, M = N = 32



(a) 1 Gbps Homogeneous Network (b) 10 Gbps Homogeneous Network (c) 1/10 Gbps Heterogeneous Network

Figure 5: Energy Efficiency Performance, M = N = 128

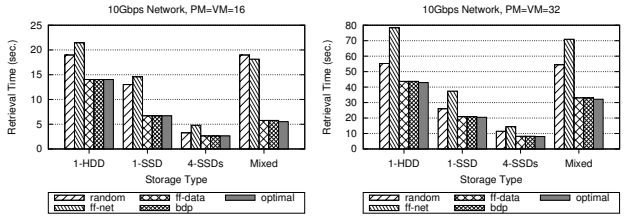


(a) 1 Gbps Homogeneous Network (b) 10 Gbps Homogeneous Network (c) 1/10 Gbps Heterogeneous Network

Figure 6: Energy Efficiency Performance, M = N = 512

clusters and the cloud. Our 10 Gbps network configuration is a good representation of this case, where all storage types provide a lower throughput than the available network bandwidth. In this case, the proposed *bdp* algorithm consistently achieves the best performance in both data retrieval and energy efficiency, as can be observed from Figures 1(b), 2(b), 3(b), and Figures 4(b), 5(b), 6(b) for retrieval time and energy efficiency, respectively.

6.2.3 Optimal Retrieval Performance. Finally, we compare the data retrieval performance of the heuristics with the *optimal* algorithm that guarantees the minimum data retrieval time. Figure 7 shows this comparison for the 10 Gbps homogeneous network case, with 16 (Figure 7(a)) and 32 (Figure 7(b)) physical machines. In three storage configurations (1-HDD, 1-SSD, and 4-SSDs) out of eight, the proposed heuristics achieved the optimal data retrieval value, and in the other five storage configurations, their performance was within 5% of optimal. These results, being close to optimal values, clearly indicate the superior quality of the data retrieval schedules determined by the proposed heuristics.



(a) M = N = 16 (b) M = N = 32

Figure 7: Retrieval Performance Compared with Optimal

7 CONCLUSION

In this paper, we formally defined and formulated Big Data aware virtual machine Placement (BDP) problem and solved it using linear programming techniques. In addition, two low-cost heuristics were proposed for efficient big data processing in the cloud that consider both the data retrieval time of large datasets and energy

consumption of the cloud infrastructures. In our evaluation, the proposed heuristics achieved a data retrieval performance within 5% of the optimal value. Furthermore, the energy efficiency of the proposed heuristics also outperformed popular energy-aware VM placement heuristics in the cases where the storage subsystem was the cause of the bottleneck in data transfer. As high-speed networking interconnects of 10/40/100 Gbps become more common in private clusters and cloud infrastructures, even new high performance NVMe storage solutions cannot keep up with the available network bandwidth. Therefore, we believe that the proposed heuristics can provide a tremendous value for big data processing in the cloud by reducing both data analysis times and energy consumption.

ACKNOWLEDGMENTS

This research was supported in part by the U.S. National Science Foundation (NSF) under grant CNS-1657296.

REFERENCES

[1] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.
[2] K. Shvachko, Hairong Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10, May 2010.
[3] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. Disk-locality in datacenter computing considered irrelevant. In *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems*, HotOS'11, pages 12–12, Berkeley, CA, USA, 2011. USENIX Association.
[4] White Paper. *NVMe SSD 960 PRO/EVO*, December 2016.
[5] Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, and Samee Ullah Khan. The rise of "big data" on cloud computing. *Inf. Syst.*, 47(C):98–115, January 2015.
[6] Domenico Talia. Clouds for scalable big data analytics. *Computer*, 46(5):98–101, May 2013.
[7] Suraj Pandey and Surya Nepal. Cloud computing and scientific applications âĂŤ big data, scalable analytics, and beyond. *Future Generation Computer Systems*, 29(7):1774 – 1776, 2013.
[8] Fabio Lopez Pires and Benjamin Baran. Virtual machine placement literature review. *CoRR*, abs/1506.01509, 2015.
[9] Chetan S. Rao, Jeffrey John Geevarghese, and Karthik Rajan. Improved approximation bounds for vector bin packing. *CoRR*, abs/1007.1345, 2010.
[10] Rina Panigrahy, Kunal Talwar, Lincoln Uyeda, and Udi Wieder. Heuristics for vector bin packing. January 2011.
[11] L. T. Chen and D. Rotem. Optimal response time retrieval of replicated data. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 36–44, 1994.
[12] L. R. Ford and D. R. Fulkerson. Maximal Flow through a Network. *Canadian Journal of Mathematics*, 8:399–404, 1956.
[13] Nihat Altiparmak and A. S. Tosun. Generalized optimal response time retrieval of replicated data from storage arrays. *ACM Transactions on Storage*, 9(2):5:1–5:36, July 2013.
[14] Nihat Altiparmak and A. S. Tosun. Integrated maximum flow algorithm for optimal response time retrieval of replicated data. In *41st International Conference on Parallel Processing (ICPP 2012)*, Pittsburgh, Pennsylvania, September 2012.
[15] Nihat Altiparmak and Ali Saman Tosun. Continuous retrieval of replicated data from heterogeneous storage arrays. In *22nd IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2014)*, Paris, France, September 2014.
[16] Nihat Altiparmak and Ali Saman Tosun. Multithreaded maximum flow based optimal replica selection algorithm for heterogeneous storage architectures. *IEEE Transactions on Computers*, PP(99):1–1, 2015.
[17] S. W. Son, Samuel Lang, R. Latham, Robert B. Ross, and Rajeev Thakur. Reliable mpi-io through layout-aware replication. In *Proc. 7th IEEE Int'l Workshop on Storage Network Architecture and Parallel I/O (SNAPI 2011)*, Denver, CO, 05/2011 2011.
[18] Kristina Chodorow and Michael Dirolf. *MongoDB: The Definitive Guide.* O'Reilly Media, Inc., 1st edition, 2010.
[19] William H. Tetzlaff and Robert Flynn. *Block allocation in video servers for availability and throughput.* IBM US Research Centers, 1996.
[20] Jose Renato Santos, Richard R. Muntz, and Berthier Ribeiro-Neto. Comparing random data allocation and data striping in multimedia servers. SIGMETRICS '00, pages 44–55, New York, NY, USA, 2000. ACM.
[21] Richard Muntz, Jose Renato Santos, and Steven Berson. A parallel disk storage system for realtime multimedia applications. *International Journal of Intelligent Systems*, 13:1137–1174, 1998.
[22] Zhibo Cao and Shoubin Dong. An energy-aware heuristic framework for virtual machine consolidation in cloud computing. *J. Supercomput.*, 69(1):429–451, July 2014.
[23] D. Dong and J. Herbert. Energy efficient vm placement supported by data analytic service. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pages 648–655, May 2013.
[24] J. Dong, X. Jin, H. Wang, Y. Li, P. Zhang, and S. Cheng. Energy-saving virtual machine placement in cloud data centers. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pages 618–624, May 2013.
[25] Daochao Huang, Dong Yang, Hongke Zhang, and Lei Wu. Energy-aware virtual machine placement in data centers. In *2012 IEEE Global Communications Conference (GLOBECOM)*, pages 3243–3249, Dec 2012.
[26] Xiaoqiao Meng, Vasileios Pappas, and Li Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *Proceedings of the 29th Conference on Information Communications*, INFOCOM'10, pages 1154–1162, Piscataway, NJ, USA, 2010. IEEE Press.
[27] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang. Joint vm placement and routing for data center traffic engineering. In *2012 Proceedings IEEE INFOCOM*, pages 2876–2880, March 2012.
[28] Stefanos Georgiou, Konstantinos Tsakalozos, and Alex Delis. Exploiting network-topology awareness for vm placement in iaas clouds. In *Proceedings of the 2013 International Conference on Cloud and Green Computing*, CGC '13, pages 151–158, Washington, DC, USA, 2013. IEEE Computer Society.
[29] Weiming Shi and Bo Hong. Towards profitable virtual machine placement in the data center. In *Proceedings of the 2011 Fourth IEEE International Conference on Utility and Cloud Computing*, UCC '11, pages 138–145, Washington, DC, USA, 2011. IEEE Computer Society.
[30] Wubin Li, Johan Tordsson, and Erik Elmroth. Virtual machine placement for predictable and time-constrained peak loads. In *Proceedings of the 8th International Conference on Economics of Grids, Clouds, Systems, and Services*, GECON'11, pages 120–134, Berlin, Heidelberg, 2012. Springer-Verlag.
[31] A. Gupta, L. V. KalÃĺ, D. Milojicic, P. Faraboschi, and S. M. Balle. Hpc-aware vm placement in infrastructure clouds. In *2013 IEEE International Conference on Cloud Engineering (IC2E)*, pages 11–20, March 2013.
[32] E. Bin, O. Biran, O. Boni, E. Hadad, E. K. Kolodner, Y. Moatti, and D. H. Lorenz. Guaranteeing high availability goals for virtual machine placement. In *31st International Conference on Distributed Computing Systems*, pages 700–709, June 2011.
[33] J. T. Piao and J. Yan. A network-aware virtual machine placement and migration approach in cloud computing. In *2010 Ninth International Conference on Grid and Cloud Computing*, pages 87–92, Nov 2010.
[34] K. Zamanifar, N. Nasri, and M. H. Nadimi-Shahraki. Data-aware virtual machine placement and rate allocation in cloud environment. In *2012 Second International Conference on Advanced Computing Communication Technologies*, pages 357–360, Jan 2012.
[35] M. Alicherry and T. V. Lakshman. Optimizing data access latencies in cloud systems by intelligent virtual machine placement. In *2013 Proceedings IEEE INFOCOM*, pages 647–655, April 2013.
[36] J. J. Kuo, H. H. Yang, and M. J. Tsai. Optimal approximation algorithm of virtual machine placement for data latency minimization in cloud systems. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 1303–1311, April 2014.
[37] Rainer E. Burkard and Eranda Çela. *Linear Assignment Problems and Extensions*, pages 75–149. Springer US, Boston, MA, 1999.
[38] R M Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, 40(4):85–103, 1972.
[39] Rina Panigrahy, Kunal Talwar, Lincoln Uyeda, and Udi Wieder. Heuristics for vector bin packing. January 2011.
[40] V Vazirani. Approximation algorithms springer-verlag. *New York*, 2001.
[41] Linux man pages. *mdadm - manage MD devices aka Linux Software RAID.* https://linux.die.net/man/8/mdadm.
[42] Linux man pages. *dd - convert and copy a file.* https://linux.die.net/man/1/dd.
[43] Linux man pages. *nc - arbitrary TCP and UDP connections and listens.* https://linux.die.net/man/1/nc.
[44] IBM ILOG CPLEX. Optimization studio for academics: High-performance software for mathematical programming and optimization. http://www.ilog.com/products/cplex/.
[45] Nihat Altiparmak and A. S. Tosun. Equivalent disk allocations. *IEEE Transactions on Parallel and Distributed Systems*, 23(3):538–546, March 2012.
[46] Amazon. *Amazon EC2 VM Instance Types*, 2017. https://aws.amazon.com/ec2/instance-types/.