Optimizing Data Access Latencies in Cloud Systems by Intelligent Virtual Machine Placement

Mansoor Alicherry
Bell Labs India, Alcatel-Lucent
Bangalore, India

T.V. Lakshman
Bell Labs, Alcatel-Lucent
New Jersey, USA

***Abstract*—Many cloud applications are data intensive requiring the processing of large data sets and the MapReduce/Hadoop architecture has become the de facto processing framework for these applications. Large data sets are stored in data nodes in the cloud which are typically SAN or NAS devices. Cloud applications process these data sets using a large number of application virtual machines (VMs), with the total completion time being an important performance metric. There are many factors that affect the total completion time of the processing task such as the load on the individual servers, the task scheduling mechanism, communication and data access bottlenecks, etc. One dominating factor that affects completion times for data intensive applications is the access latencies from processing nodes to data nodes. Ideally, one would like to keep all data access local to minimize access latency but this is often not possible due to the size of the data sets, capacity constraints in processing nodes which constrain VMs from being placed in their ideal location and so on. When it is not possible to keep all data access local, one would like to optimize the placement of VMs so that the impact of data access latencies on completion times is minimized. We address this problem of optimized VM placement – given the location of the data sets, we need to determine the locations for placing the VMs so as to minimize data access latencies while satisfying system constraints. We present optimal algorithms for determining the VM locations satisfying various constraints and with objectives that capture natural tradeoffs between minimizing latencies and incurring bandwidth costs. We also consider the problem of incorporating inter-VM latency constraints. In this case, the associated location problem is NP-hard with no effective approximation within a factor of $2 - \epsilon$ for any $\epsilon > 0$. We discuss an effective heuristic for this case and evaluate by simulation the impact of the various tradeoffs in the optimization objectives.**

I. Introduction

With the immense proliferation of cloud-hosted applications, there has been corresponding interest in optimizing cloud systems to meet the performance and cost objectives of various classes of applications. Many popular cloud applications are data intensive with applications ranging from those which use the MapReduce framework for processing several petabytes a day [6] to web and video applications that tradeoff computing and storage [8]. For many of these data-intensive applications, the Map-Reduce/Hadoop processing paradigm has become the method of choice for executing these applications. The Hadoop[1] system can partition computations and data over thousands of servers and storage nodes in a cloud system, thereby immensely speeding up completion times for the application.

When computation and data are spread over a large number of nodes, with completion time being an important metric for the application, then an important problem is the effective placement of computation and data to achieve fast completion times. A poor placement may lead to large data access latencies that result in increased completion times. For instance, if the computation VMs and corresponding data nodes are placed on different racks then the typical oversubsciption of aggregation layer links in a data center leads to potential network bottlenecks that can cause data access latencies. Ideally, it would be preferable to keep all data access local and systems such as Hadoop try to accomplish this by reducing the amount of remote data access. The Hadoop task scheduler, for instance, attempts to match a task with a processing node that has data locally. When such a match is not feasible, the system tries to find nodes with data in the same rack. If this too fails, only then are off-rack nodes chosen.

In general, it will not be possible to keep most data accesses local. To keep data accesses local one must place VMs locally to where the data is stored or move the data to where VMs can be placed. Neither option is always feasible. It may not be possible to place VMs locally to nodes that store data. Large data sets for a particular application may be stored in NAS or SAN devices that may be located in a only a subset of the nodes in a cloud system. Moving the VMs to these data nodes or moving the data to computation nodes may be infeasible due to capacity constraints. In these cases, irrespective of whether the data-intensive applications use the Map-Reduce/Hadoop framework or not, it is important to carefully place computation (VM) nodes so that data access latencies are minimized. This optimized placement is particularly important in geographically distributed clouds consisting of a large number of relatively small computation and storage nodes. For such systems, the large differences in access latencies for different data nodes can lead to large increases in completion times if VMs are not optimally placed.

The focus of this paper is on the optimized placement of virtual machines to minimize data access latencies. We discuss the problem in detail in the next section and describe the different possible optimization objectives that tradeoff performance for bandwidth costs. The main contribution of the paper is the algorithms that we present in Section III for the optimal

978-1-4673-5946-7/13/$31.00 ©2013 IEEE

placement of virtual machines. Section IV shows that adding inter-VM distance constraints makes the problem NP-hard and difficult to approximate as well. Effective heuristics for this case are presented. We also study by extensive simulations, in Section V the various tradeoffs discussed in Section II. Section VI discusses related work and concluding remarks are in Section VII.

II. Problem Description

We assume a distributed cloud environment [10] where there are a large number of small datacenters dispersed in an area. However, the solutions in the paper are applicable to large centralized cloud settings as well. There are two types of resources inside the datacenter: compute nodes and storage nodes. A user's data is stored in a subset of the storage nodes. The user wants to perform certain computation on their data. The user wants to allocate certain number of compute resources (VMs) for this computation. We assume one VM accesses data from one storage node and vice versa. For example, in Amazon EC2 environment the compute node mounts the data node as a volume and access the volume. It is not possible to mount the same volume in multiple VMs simultaneously. However, our algorithms can also handle multiple VMs accessing the same data node or single VM accessing multiple data nodes by creating multiple instances of data nodes or consolidating the data node instances respectively.

In our problem setting, location of the data nodes are fixed, and allocated apriori. Different compute nodes (Racks/blades/CPUs) have certain number of VMs available. For each compute node, we know the access latency or bandwidth to the data nodes. Our goal is to select the VMs that minimize the latency of access to the data, or minimize the total bandwidth consumed for the data transfer or maximize the available bandwidth to the data. Each of the selected VMs is assigned to the data node which contains the data that will be processed by the VM.

In the rest of the paper, we will describe the algorithms in terms of minimizing the latency. We assume that the datacenter has predictable latency between the nodes, and the latency and bandwidth usage are proportional. For the problems, where we want to minimize the bandwidth usage, we can directly use the bandwidth usage numbers in the algorithms without any modification. If the metric we want to use is maximizing available bandwidth, we can create a minimization problem by replacing the link available bandwidth with difference of maximum available bandwidth over all the links and the available bandwidth of that link.

There are several possible optimization objectives which reflect the latency and bandwidth cost tradeoffs. Depending on the application, we can use any of the following objectives while choosing the VMs for the data nodes:

1) **Minimize total (or average) access time/bandwidth:** Here we minimize the sum of the access latencies between the data node and the corresponding VMs. Minimizing this metric will reduce the overall bandwidth cost of running the job. If the latencies of accessing the VMs are skewed, the solution may involve few assignments that have large latencies compared to rest of the assignments. In those cases the performance of the job may be adversely affected by these high latency assignments.
2) **Minimize maximum access time:** Here we minimize the maximum latency between any data node and the corresponding VM. This objective is used for jobs where the performance of the application is of at most importance. The solution may lead to very high total access time/bandwidth.
3) **Minimize the total access time/bandwidth within an access time threshold:** Here the goal is to minimize the sum of latencies (or bandwidth) with the constraint that all the latencies between a data node and the corresponding VM has to be less than a threshold. This is a compromise between the first two objectives; here we find a solution that has minimum cost while guaranteeing some performance.
4) **Minimize maximum access time within a total access time/bandwidth threshold:** This is the complement of the above problem. We are given a total access time threshold, and wants to minimize the maximum access time of any data node to its corresponding VM, subject to that total access time threshold. This is also a compromise between the first two objectives; here we find the solution that has maximum performance with a cost budget.

A. Inter VM constraints

In cloud applications like map-reduce, it is not sufficient that the VMs running the jobs are close to the data. There might be communication between these VMs itself. Hence, for better performance, the chosen VMs have to be close to each other. The user may specify that the VMs that are selected have to be within a fixed distance of each other.

The objectives mentioned previously does not take the inter-VM distance into consideration. This may lead to solution that assigns VMs, which are farther apart in the datacenter or distributed cloud, to the data nodes. The performance of the application may be adversely affected when these VMs have to exchange data, due to high latency or low bandwidth available between the VMs.

In this paper, we also consider the problem of assigning VMs to the data nodes that takes into account of inter-VM communication of the application. We represent the additional constraints in the form of inter-VM distance or latency. We provide algorithms for each of the above optimization objectives with additional constraint on the maximum (threshold) inter-VM communication distance/latency for the VMs assigned to the data nodes.

III. Algorithms with no inter-VM constraints

In this section, we provide algorithms for the data-aware VM placement problem. Our solutions are based on algorithms

for assignment problems. The problems involving inter-VM constraints are NP-hard and we provide heuristic solutions of those problems in the next section.

Let $D_1, D_2, \ldots, D_k$ be the data nodes that require VMs for processing the data. Let $V_1, V_2, \ldots, V_m$ be the available virtual machines that can be assigned for processing these data. Let d_{ij}, $i \in \{1, \ldots k\}$, $j \in \{1, \ldots m\}$ represent the distance, latency or the bandwidth cost between the data D_i and VM V_j. If a VM cannot be assigned to a data node, then we set the corresponding d_{ij} to ∞.

Our goal is to find an assignment of data nodes to VMs such that no two data nodes are assigned to the same VM and vice versa, and all the data nodes are assigned to VMs. Let x_{ij} be the binary variable, set to 1 if and only if data node D_i is assigned to VM V_j.

We can treat the VM placement problem as a classic linear sum assignment problem [5]. To transform the VM placement problem into a classic assignment problem, we first make the cardinality of data nodes and VMs the same. If the number of VMs is less than the number of data nodes, then there is no solution to the VM assignment problem, since every data node requires a VM to process the data. If the number of VMs is more than the number of data nodes, we add dummy data nodes to make their cardinalities the same. We set the cost of assigning VMs to the dummy data nodes as 0. In the transformed problem, let n be the number of data nodes and VMs.

The assignment problem is to minimize the cost of the assignment subject to the constraint that each object on either side of the assignment appears exactly once. The assignment problem can be represented as the following 0-1 linear program:

Minimize $\sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$
Subject to
$\sum_{i=1}^{n} x_{i,j} = 1$ for all $j = 1 \ldots n$
$\sum_{j=1}^{n} x_{i,j} = 1$ for all $i = 1 \ldots n$
$x_{ij} \in \{0, 1\}$ for all $i = 1 \ldots n$ and $j = 1 \ldots n$

The constraint matrix is totally unimodular. We can relax the conditions $x_{ij} \in \{0, 1\}$ to $x_{ij} \geq 0$ and solve the corresponding linear program.

The dual of the above program is as follows. There are dual variables associated with each data nodes and VMs. Let's call them u_i and v_j respectively. The dual program is:

Maximize $\sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j$
Subject to
$u_i + v_j \leq d_{ij}$ for all $i = \ldots n$ and $j = 1 \ldots n$

Hence the problem becomes that of assigning weights to data nodes and virtual machines, such that their sum is maximized subject to the cost constraint.

There are a large number of algorithms to solve the linear assignment problem. For this paper, we use the classic *Hungarian algorithm* [5] to solve this problem. This is a primal-dual algorithm.

Edge weight transformation

In the description of the problems and solution so far, we assume that each VMs access the same amount of data from the data nodes. Hence we measure the cost of the solution in terms of latency of the VMs to the data nodes. In practice, the amount of data that will be accessed by the VMs from the data nodes will be different from each data nodes. We can account for the non-uniform data access by the VMs by modifying the cost of the links between the data nodes and the VMs.

Let $p_1, p_2, \ldots, p_k$ be the amount of data that need to be processed from the data nodes $D_1, D_2, \ldots, D_k$ respectively. The total bandwidth required for accessing data from a data node by the VM assigned to the data node is proportional to the amount of data that will be accessed as well as the distance (or latency) of the data node from the VM. Hence, the cost of accessing the data from node D_i by VM V_j is $d_{ij} p_i$. Given an assignment x_{ij} between the data nodes and the VMs, the total cost of access is:
$\sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij} p_i$
All the other constraints of the original program remain unchanged. Hence, for optimizing for the uneven access of data, we use the same solution as the uniform weight, but with modified edge costs of $d_{ij} p_i$

Example: Figure 1(a) shows an example of a VM assignment problem, where there are 3 data nodes of size 20 GB, 10 GB and 8 GB. There are 4 VMs to choose from. The labels on the links show the latency or bandwidth costs. If the assigned VMs access all the data that is present in the corresponding data node, then we can modify the problem to the one in Figure 1(b) to accurately reflect the costs of data access. Here each link cost of the original problem is multiplied with the data size of the incident data node.

Multiple VMs accessing the data

In some application scenarios, a data node may need to be processed by multiple virtual machines. For example, if the amount of data at a node is significantly larger than other data nodes, then the user may request that the larger data be processed by multiple virtual machines. We transform such a problem into our model of one VM per data node by creating multiple instance of the data node. Note that creating multiple instances is done only as an input to the algorithm; but it is not done to the actual data present in the original data node. Multiple VMs output by the algorithm is then assigned to the same data node. We can also use the edge transformation described previously to assign different proportions of the data from the data node to different VMs.

Similarly, a VM may be required to access data from multiple data nodes. For example, in a map-reduce job, data for a reduce node may come from multiple map tasks that may store the data in different data nodes. This may also be done to reduce the total number of VMs to run a job as the dollar cost of accessing the cloud is proportional to number of VMs. In those cases, we combine the data nodes into one logical data node while giving them as input to the algorithm. The

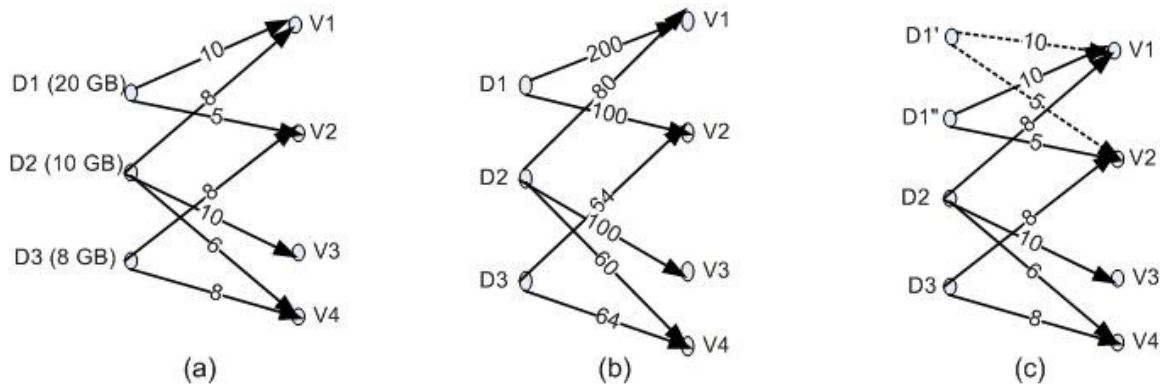

Fig. 1. Example edge weight transformation

cost of the links from this logical data node to a VM is taken as the average of the cost between the VM and the individual data nodes. Here again, we can use the weighted average for the link costs if the amount of data access by VMs is different from each of the data nodes.

Example: Figure 1(c) shows the case where two VMs are needed to process the data from data node D_1. Here we create two instances of D_1 named D_1' and D_1". Links from the original data node to the VMs are replicated in both of these instances.

A. Algorithms

In this section we describe algorithms for each of the problems described in Section II.

1) Minimize total (or average) access time/bandwidth: In this case we want to minimize the total access time. The objective of the problem is to minimize $\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij}$. We use the standard assignment algorithm discussed at the beginning of the section to solve this problem.

2) Minimize maximum access time: In this case, our goal is to minimize the maximum latency required by any VM to access the corresponding data node. This can be formulated as the following program:

Minimize $\max_{ij} d_{ij}x_{ij}$
Subject to
$\sum_i x_{i,j} = 1$ for all $j = 1 \ldots n$
$\sum_j x_{i,j} = 1$ for all $i = 1 \ldots n$
$x_{ij} \in \{0, 1\}$ for all $i = 1 \ldots n$ and $j = 1 \ldots n$

This problem is an instance of linear bottleneck assignment problem (LBAP) [5]. LBAP is a linear assignment problem whose objective is to minimize the maximum cost among the individual assignment. One of the solution techniques to solve LBAP is to use a *threshold* algorithm. The algorithm fixes a threshold for the solution. It constructs a bipartite graph whose edges are the edges from the original graph whose lengths are below the threshold. Any matching on this graph is the solution to the original with the objective value of the threshold. Optimal value of the threshold can be found by doing a binary search on the threshold that has complete matching. There are also other algorithms based on augmenting paths [5].

3) Minimize the total access time/bandwidth within an access time threshold: In this problem, we want to minimize the total cost of the assignment, subject to the constraint that none of the edge weights in the assignment can be more than a given threshold. This problem can be transformed into the total access time minimization problem (problem 1), by removing all the edges that are above the threshold, or by giving the edges that are above the threshold the cost of infinity (if the algorithm requires complete bipartite graph). In fact, it is also possible for the user to give individual thresholds to each of the data node. For example, if the data in some data nodes are important or takes more time to process, those data nodes may be assigned a lower threshold. In those cases, we will remove the edges incident on the data nodes which are more than the corresponding thresholds from the bipartite graph. Then we run the cost minimization algorithm on the modified graph.

4) Minimize the maximum access time within a total access time/bandwidth threshold: In this problem, like the second problem, we want to minimize the maximum of all the access times. But, we do not want the total access time to go above a threshold (budget) value.

Let's consider the reverse problem of minimizing the total access time subject to a threshold on maximum access time on any link. As stated above, we can solve this problem by pruning the links that does not satisfy the link threshold constraint. As the threshold increases, only new links gets added to the graph, and the existing links are not removed. Hence, the total access time decreases as the threshold increases. To solve the problem of minimizing the maximum access time within the total access time threshold, we do the following. Like the threshold algorithm, perform a binary search on the access time of the links. In each search step, the value of the solution is the minimum total access latency for an instance where all the links with latency above the given access time are pruned. We take the solution that has the lowest maximum link access time, whose total access time is below the given budget.

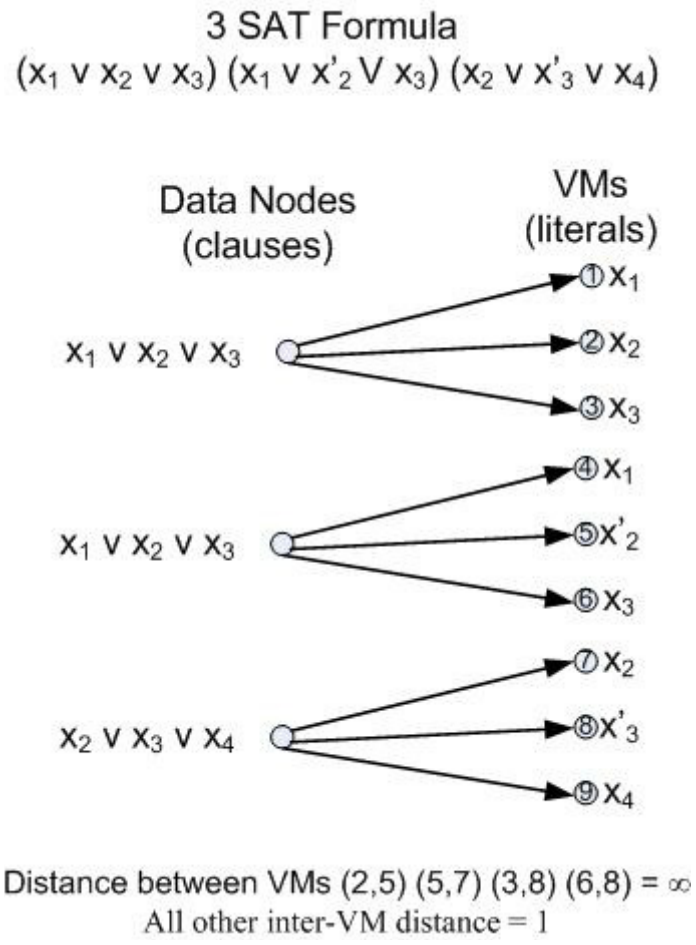

Fig. 2. NP-hardness reduction from 3-SAT

IV. Algorithms for inter-VM constraint

In this section, we consider the allocation of VMs to data nodes, where user gives additional constraints of distance between the VMs assigned to the data nodes. First, we prove that the problem is NP-hard, with no possible approximation for arbitrary inter VM distance. If the inter-VM distance satisfies triangle inequality, then the problem cannot be approximated within a factor of $2-\epsilon$ for any $\epsilon > 0$. Then we give heuristic algorithms for the problem, which provides a generic framework for using all the algorithms mentioned in previous section.

A. NP-hardness

We prove the NP-hardness by reduction from 3-satisfiability (3-SAT) problem. Given a 3 SAT instance with n variables and m clauses, create the VM assignment instance as follows. There are m data nodes, each corresponding to a clause. There are $3m$ VMs corresponding to the literals of the clauses. A data node corresponding to a clause can be assigned to any of VMs corresponding to its literals and the latency between this data node and the VMs are 1. Latency between a data node corresponding to a clause and the VMs corresponding to another clause is $r > 1$. The distance between VMs corresponding to any literal and its negation is $s > 1$. The distance between any other VMs is 1.

We prove that an assignment of the VMs to the data nodes whose inter-VM distance is 1 and total latency m corresponds to a valid 3 SAT assignment and vice versa.

If there is a solution to the 3 SAT instance, we can find a solution for the VM placement as follows. For each of the data nodes corresponding to a clause, assign a VM from its corresponding literal that has the satisfying assignment. The latency of this assignment will be 1 for each of the data nodes. Furthermore, the distance between any two VMs that are assigned to the data nodes is 1. This is because, all the VMs that are assigned to the data nodes corresponds to literals in a satisfying assignment. Only VMs corresponding to a literal and its complements has inter-VM distance more than 1, and any satisfying assignment cannot have both a variable and its compliment set to one. The latency of this assignment is 1 for each data node, giving the total latency of m, and the inter-VM distance in the assignment is 1.

Similarly, if there is a solution for the VM placement, whose total latency is m and inter-VM is 1, then the corresponding 3 SAT instance is satisfiable. Given such a solution to the VM placement, we can create the solution for 3 SAT instance as follows. Since the latency of any data node to any VM is at least one and there are m data nodes, the latency to each of the assigned VM from the corresponding data node has to be exactly 1. Hence, each of the assignment of the data node to VM in the solution corresponds to assignment of the corresponding clause to one of its literals. Furthermore, since the distance between any two VMs that are assigned to the data nodes is 1, none of those VMs corresponds to a literal and its compliment. Otherwise, the distance between those VMs would have been $s > 1$. Hence, to create a satisfying assignment, assign true to all the literals that has a corresponding VM assigned to a data node. If no VMs corresponding to a variable or its complement are assigned to a data node, then the variable can take either true or false without affecting the satisfiability of the clauses.

Any solution that corresponds to a non-satisfying assignment of the 3-SAT will have: either an assignment of a data node to a VM whose latency is r which corresponds to an unsatisfiable clause; or two VMs in the assignment whose inter-VM distance is s, which corresponds to both a variable and its compliment assigned the value of true; or both.

Note that we did not make any assumption on the value of r and s other than they being greater than 1. Setting these values to a large quantity shows the inapproximability of the VM assignment problem with inter-VM constraints; the problem cannot be approximated with in any factor.

In many cases, the distance between the VMs follows triangle inequalities. In our reduction, to satisfy triangle inequality, the value of s has to be at most 2. In that case the inter-VM distance between an optimal and non-optimal solution will be 1 and 2, giving an optimality gap of 2. Hence, the problem is $2-\epsilon$ inapproximable when the inter-VM distances follows

triangle inequality.

Example: Figure 2 shows an example of a 3-SAT instance with 3 clauses and 4 variables, and corresponding VM assignment problem. Here the value of both r and s are set to ∞.

B. Algorithm for Inter-VM constraint

In this section, we provide algorithms for solving all the four problems mentioned in section II with additional constraints on the inter-VM distance. We assume that distance between the VMs follows triangle inequality.

Our algorithm is based on the following observation. Let t be the inter-VM distance threshold. Consider a graph G_v whose set of nodes corresponds to VMs and there is an edge between the nodes if the distance between the corresponding VMs is less than t. The nodes in G_v, which corresponds to VMs in an assignment that satisfies the inter-VM distance constraint, forms a clique. The basic idea of our algorithm is the following: Given a partial assignment of data nodes to VMs (i.e. only some of the data nodes are assigned VMs), then for incrementally assigning more VMs to unassigned data nodes, we can consider only the VMs that forms a clique with the currently assigned VMs.

Algorithm 1 Assignment-with-inter-VM$(G, t, \mathcal{P}, \mathcal{A}_\mathcal{P})$

Input: $G = (D, V)$: bipartite graph with data nodes and VMs
t: Threshold for inter VM distance
$\mathcal{P}$: Assignment problem without the inter-VM constraint
$\mathcal{A}_\mathcal{P}$: Algorithm for $\mathcal{P}$
Output: Solution for problem $\mathcal{P}$ with additional inter-VM distance constraint on Graph G
$\mathcal{C} \leftarrow$ Set of candidate cliques of VMs in G with threshold t
$bestSolution \leftarrow \phi$
for all $C \in \mathcal{C}$ **do**
  Create problem instance $G' = (D, C)$ for problem $\mathcal{P}$ that contains all the data nodes and only VMs in C
  $currentSolution \leftarrow$ result of running $\mathcal{A}_\mathcal{P}$ for problem $\mathcal{P}$ with input G'
  if $currentSolution$ better than $bestSolution$ **then**
    $bestSolution \leftarrow currentSolution$
  end if
end for
return $bestSolution$

The general framework of the algorithm for problem $\mathcal{P}$ with additional inter-VM distance constraint is given in Algorithm 1. Let $\mathcal{A}_\mathcal{P}$ be the algorithm for the problem $\mathcal{P}$ without the inter-VM distance constraints, similar to the one described in Section III. We select a set of VMs $\mathcal{V}$ that form a clique based on the inter-VM distance threshold t. Note that the clique $\mathcal{V}$ is not the maximum clique, as we cannot find maximum clique in polynomial time. We invoke the algorithm $\mathcal{A}_\mathcal{P}$ on the input consisting of all the data nodes and set of VMs $\mathcal{V}$ that form a clique. This process is repeated for different set of cliques, and we select the best result.

When the inter-VM distances follows the triangle inequality, we can select the clique with threshold at most t as follows. We start with VM v, and add it to the clique. Now add all the VMs that are at a distance at most $t/2$ from v to the clique. The set formed is a clique with threshold distance t since the distance between any two VMs in the set is less than the sum of the distance between those VMs and v, due to triangle inequality. We can also add additional VM to the clique as follows: consider one VM at a time that is not part of the clique. If the distance to that VM from all the VMs that are already present in the clique is less than the threshold, then we add that VM to the clique. The algorithm is presented in Algorithm 2.

We create cliques with each of the VM as the starting VM, and run the algorithm 1 with those cliques. We take the solution with the minimum cost.

Algorithm 2 VM-clique(G, v, t)

Input: $G = (D, V)$: bipartite graph with data nodes and VMs
v: Initial VM added to the clique
t: Threshold for inter-VM distance
Output: A maximal clique centered at node v
$C \leftarrow \{v\}$
for all $u \in V$ **do**
  if $dist(u, v) \leq t/2$ **then**
    $C \leftarrow C \cup \{u\}$
  end if
end for
for all $u \in V - C$ **do**
  flag $\leftarrow$ true
  for all $w \in C$ **do**
    if $dist(u, w) > t$ **then**
      flag $\leftarrow$ false
      break
    end if
  end for
  if flag = true **then**
    $C \leftarrow C \cup \{u\}$
  end if
end for
return C

V. Simulation Results

In this section we evaluate the performance of virtual machine placement for low latency data access under different constraints. First we study the performance without inter-VM distance constraint and then study by adding that constraint.

For our simulations, we create demand graphs as follows. We assume a datacenter setup with 1024 racks, which is organized in a hierarchical manner. Nodes that belong to racks that are in blocks of 16 (i.e. 0-15, 15-31 etc) can communicate with each other using a single switch. Nodes that belong to the racks that are in the same blocks of 64, but not in the

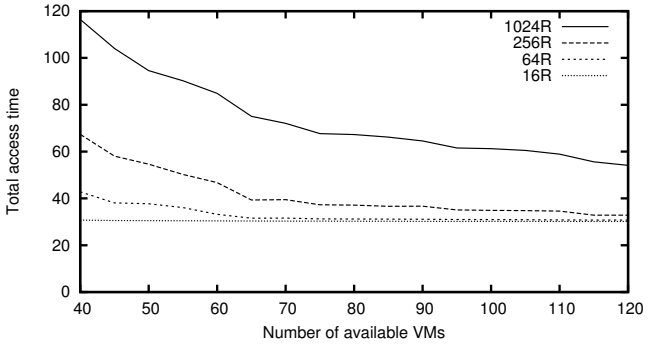

Fig. 3. Total access time for different number of VMs for a 40 data node assignment that minimizes the total access time

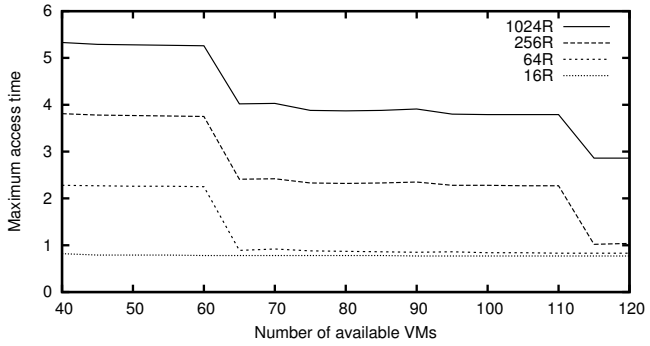

Fig. 4. Maximum access time vs number of VMs for a 40 data node allocation that minimizes the maximum access time

same blocks of 16 can communicate with each other using 3 switches. Similarly nodes that belong to racks that are in blocks of 256 can communicate with each other using 5 switches. If the node belongs to racks that are not in blocks of 256, then they communicate using 7 switches. To create the demand matrix, we first create required number of data nodes and VMs and randomly assign them to one of the racks. While selecting the racks for the data nodes and VMs, we may also restrict it to a subset of racks (e.g. one of the first 256 racks). The latency between a data node and a VM is taken as random between $0.75 - 1.25\times$ the number of switches between them.

We conducted various experiments where the number of data nodes varied from 10 nodes to 80 nodes in multiples of 10. The number of VMs available also varied from 10 to 120 VMs in multiples of 5. Both data nodes and VMs were placed at random in first k racks for $k = 16, 64, 256$ and 1024. In this paper, we present the results only for 40 data nodes. The results were similar for other number of data nodes. Each run of the experiment was done for 20 times with different random seeds for the latency between the data nodes and the VMs, and we report the average of these experiments.

A. Minimizing total access time

First we conduct experiments to study the effect of number of available VMs on the total access time. Here we varied the number of available VMs from 40 to 120 and ran the algorithm that minimizes the total access time. Figure 3 reports the results for different number of rack sizes. The labels on the graph show the number of data nodes and the number of racks. As it can be seen, the total access time of the assignment decrease as number of VMs increases. This is because the as the number of available VMs increases, there will be more VMs closer to the data nodes. The data nodes will also be able to choose from more VMs. Figure also shows that as we restrict the data nodes and VMs to smaller number of racks, the total access time decreases. This is because smaller number of racks leads to the data node and VMs to be present close to each other, leading to smaller latencies between them.

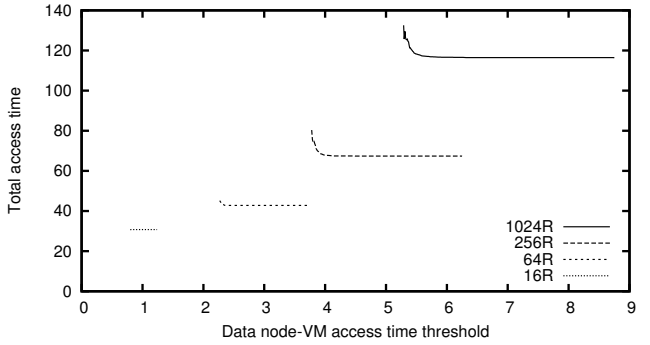

Fig. 5. Total access time vs access time threshold for a 40 data node allocation that minimizes total the total access time

B. Minimizing maximum access time

In this experiment, we use the same setup as the previous experiment, but want to minimize the maximum access time among all the assignments. Figure 4 shows the maximum data node-VM access time for different number of available VMs for an assignment consists of 40 data nodes. The maximum access time decreases as the number of VMs increases. This is again due to more choices available for data nodes, and more VMs gets closer the data nodes as the number of VMs increases. The maximum access time also decreases as we reduce the number of racks. The reason for it is same as the one mentioned in the previous experiment.

C. Minimizing total access time within an access time threshold

Now we study the effect of placing a threshold on maximum access time for the data access by VMs on the total access time. In this experiment, there were 40 data nodes and 40 VMs. We prune the edges in the data node-VM bipartite graph that exceeds the threshold, and run the total access time minimization algorithm. Figure 5 shows the total access time for different access time thresholds. As the access time threshold increases the total access time decreases. This is because, as we increase the access time threshold, we add more links to the bipartite graph. Hence, there are more choices for the assignment, which could lead to smaller total access time. We also note that after a while, as the threshold increases, the

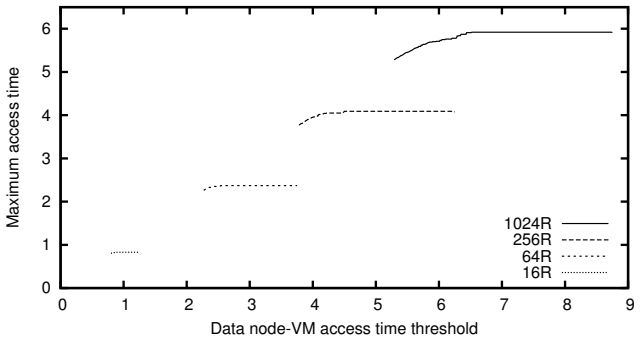

Fig. 6. Maximum access time vs access time threshold for a 40 data node allocation that minimizes the total access time

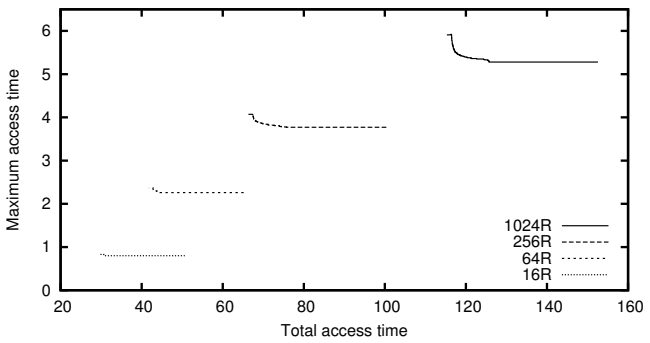

Fig. 7. Maximum access time vs total access time for a 40 data node allocation that minimize the maximum access time within total latency

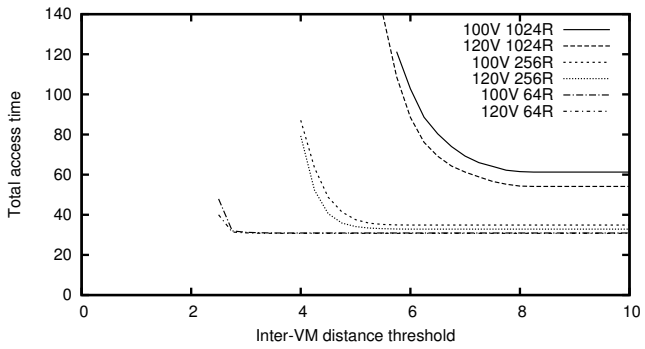

Fig. 8. Total access time vs inter-VM latency threshold for a 40 data node allocation with 100 and 120 available VMs

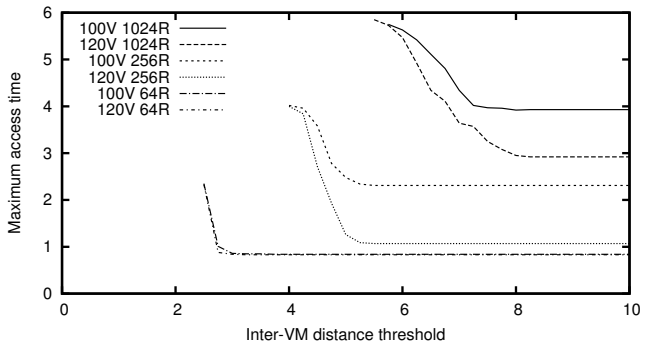

Fig. 9. Maximum access time vs inter-VM latency threshold for a 40 data node allocation with 100 and 120 available VMs that minimizes total latency

total cost remains the same. This is because those newly added links have higher cost and do not participate in the assignment. Figure 6 shows the maximum access time in the assignment for different access threshold for the same experiment. The maximum latency increases as the threshold increases. Hence, the assignment is using some larger latency links, which can free up some of the VMs. These VMs can be assigned to closer data nodes, decreasing the overall (total) access time. Like the previous experiments, the total and maximum access time decreases as the number of racks used for the VMs and the data nodes decreases.

D. Minimizing maximum access time within a total access time threshold

Now we study the effect of limiting the total access time on the maximum access time. Figure 7 shows the maximum access time as a function of total access time, that minimizes the maximum access time subject to that total access time threshold. We can see that the maximum access time decreases, as we are willing to pay with more total access time. This is because with higher total access time, the assignment is able to make more choices that decrease the maximum access time. Maximum access time cannot be reduced indefinitely by increasing the total access time. After a point, we reach the optimum access time and further increase in the total access time budget cannot reduce the maximum access time. The graph also shows the results of restricting the number of racks, and they are similar to previous results.

E. Inter-VM latency/distance constraints

Now we study the effect of inter-VM latency threshold on the total access times of the assignment. Figure 8 shows the total access time for different inter-VM latency constraints for an allocation that minimizes the total access time. The number of available VMs was 100 and 120. These are marked with 100V and 120V respectively in the graph. Figure 9 shows the maximum access time for the same experiment. As the inter-VM latency increases, the total access time decreases. This is because there are more VMs available for assignment when the inter-VM latency increases. For the same reason, the total access time also decreases as the number of available VMs increases or number of racks decreases. It is also interesting to see that for 120 VMs (on 1024 racks), the solution was available (even though of higher cost) for a threshold, for which there was no solution with 100 VMs. This is also again due to more VMs available for assignment within the threshold inter-VM latency for 120 VM case.

VI. Related work

The need to schedule computations close to where the data is located is well recognized. Schedulers which combine data locality with classical scheduling considerations like fairness have been proposed. In Quincy[7] it is noted that fairness and locality requirements often impose conflicting constraints. To

enhance the chances of local access, it may be best to delay a job's executions until its ideal resources become available. However, this delay may violate fairness needs which require the system to execute the job sooner. The Quincy scheduler uses a graph model with edge weights that reflect both fairness and locality needs. The scheduling problem is then mapped to a min-cost flow problem on this graph and an algorithm developed. This scheduler is envisaged to run on fine-grained time scales. The problem that we consider is an optimal assignment of VMs to locations without any of the other scheduler constraints and operating at larger timescales. In [11], a scheduling scheme called delay scheduling is proposed to balance data locality and fairness needs. By delaying execution of certain tasks with non-local data that would be eligible for execution for fairness reasons, it is shown that throughputs can be increased by a factor of 2. This again shows that data access performance is crucial to overall performance.

The impact of skewed popularity of data items is studied in [2] where it is observed that machines and racks that host popular data items become bottlenecks. This results in increased completion times. To alleviate this a scheme for judicious replication of popular data items which are frequently accessed is proposed.

Since MapReduce/Hadoop executes many tasks in parallel, the job completion time is delayed by stragglers or late finishing tasks. In a heterogeneous environment, late finishing tasks may be due to tasks running on a slower processor or more loaded processor, mismatched configuration, etc. Another reason, more pertinent to the problem considered in this paper, is poor VM placement that may lead to higher data access latencies and hence slower completion time for that task. To mitigate the effect of lagging tasks, MapReduce can run a speculative copy or backup task for a task that is making slow progress[6]. In Hadoop, speculative tasks are started by comparing each task's progress to the overall average progress. In [12], a much improved speculative task execution scheme for speeding up application completion times is presented – the main idea is to pick tasks for speculative execution that will finish farthest into the future. With the optimal assignment of VMs that we consider, the contribution of data latencies to the straggler problem is mitigated.

Another proposal to control the impact of stragglers on completion time is in [4]. Here, a judicious placement of tasks that takes into account network bottlenecks and other optimizations is used in addition to re-start of slow progressing tasks. A greedy algorithm is used for network-award task placement with the objective of minimizing the maximum data transfer time for MapReduce jobs.

In [9], it is noted that the MapReduce/Hadoop framework is tailored for tasks involving a large number of sequential read and writes. However, many applications also need auxiliary data that needs to be randomly accessed and bottlenecks in access to this data will lead to slow completion time. To mitigate this a memcached based in-memory object cache is adapted for use in conjunction with Hadoop.

Finally, in [3] it is argued that disk locality may not be relevant anymore because with 10G and 100G adapters access data access over a network is not necessarily slower than disk I/O. However, it is still to be noted that unless over subscription of links is avoided, network bottlenecks can cause high latencies for remote data access. This needs intelligent VM placement as we consider.

VII. Conclusions

Data access latencies can be a big contributor to the total completion times for data intensive cloud applications. Since making all data access local is often infeasible when working with large data sets, it is important to place computation nodes as close to the needed data as is possible within system and application constraints. Optimal placement is not only important for better performance but also for lower bandwidth usage costs for cloud applications. In this paper, we considered this problem of optimal placement of computational nodes and presented algorithms for assigning virtual machines to data nodes that minimize various latency metrics under different constraints. We considered minimizing total access time, maximum access time and a combination of both. We provided algorithms for problems with and without constraints. Our algorithms were based on classic linear assignment algorithms. We conducted extensive simulations to study the trade-off between various latency matrices and constraints.

References

[1] hadoop.apache.org. 2012.

[2] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris. Scarlett: Coping with skewed content popularity in mapreduce clusters. *Proceedings of the EuroSys conference*, 2011.

[3] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica. Disk-locality in datacenter computing considered irrelevant. *Proceedings of USENIX HotOS*, 2011.

[4] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris. Reining in the outliers in map-reduce clusters using mantri. *Proceedings of the OSDI conference*, 2010.

[5] R. E. Burkard and E. Cela. Linear assignment problems and extensions. *Handbook of Combinatorial Optimization: Supliment Volume A*, 1999.

[6] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *CACM*, 2008.

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

[8] A. Kathpal, M. Kulkarni, and A. Bakre. Analyzing compute vs. storage tradeoff for video-aware storage efficiency. *Proceedings of 4th USENIX Workshop on Hot Topics in Storage and File Systems*, 2012.

[9] J. Lin, A. Bahety, S. Konda, and S. Mahindrakar. Low-latency, high-throughput access to static global resources within the hadoop framework. *Technical Report HCIL-2009-01, University of Maryland, College Park, USA*, 2009.

[10] SCOPE Alliance. Telecom grade cloud computing. *www.scope-alliance.org*, 2011.

[11] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. *Proceedings of the EurosysI conference*, 2010.

[12] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. *Proceedings of the OSDI conference*, 2008.