VNetLab Resource Allocator

Ravi K. Chintallapudi

December 18, 2008

1 AIM

We aim to develop an online resource allocation algorithm for the interactive VNetLab project testbed. The interactive nature of the system disqualifies scheduling as an optimization technique. So the online algorithm optimizes the average rate at which networks complete on the testbed by efficient use of the resources – CPU, memory and network bandwidth. The virtual networks arrive in an online manner and once allocated, they cannot be migrated. To measure the performance of the algorithm we also developed an optimal resource allocation algorithm which provides every executing virtual network the fastest possible rate of completion by balancing the aggregate load and balancing and minimizing communication overhead across all the physical machines.

2 Testbed Model

2.1 Testbed Primitives

"VNetLab" project overlay's virtual networks on a physical network as depicted in Figure 1. Figure 1 shows a testbed with 4 physical nodes (boxes) and a 5 node virtual network overlayed on the testbed. We first identify the elements of the testbed that perform work (computation and communication) and contribute to the running time of an individual virtual network. These primitives - Intra-Node Links, Inter-Node Links and Physical Nodes, are starred in Figure 1. The Intra-Node Links are the virtual links between virtual machines which are hosted on the same physical node while Inter-Node Links are the virtual links between virtual machines which are hosted on different physical nodes.

2.2 Specification

Each node in a virtual network requires some amount of CPU-cycles per sec to execute and generates some network traffic per second (bandwidth). Also each node requires some amount of physical memory to execute. A virtual network can be specified as in Figure 3 by a set of "triples and a list" pairs that specify: (number of CPU-cycles per second each virtual machine requires, the network traffic generated per second, the amount of memory required by each virtual machine) and a list specifying the distribution of data generated by the processor on the links incident to the virtual machine (clockwise direction in figures).

The above specification is constructed totally using user supplied data. The specification can be further refined by using additional information about the virtual machine image (Is it a server image? Number of applications installed (code)?). Using the above information and depending on the physical location of each virtual machine, an estimate of communication and computation load exerted by the virtual networks on each physical machine in the testbed can be obtained and performance of virtual networks can be determined.

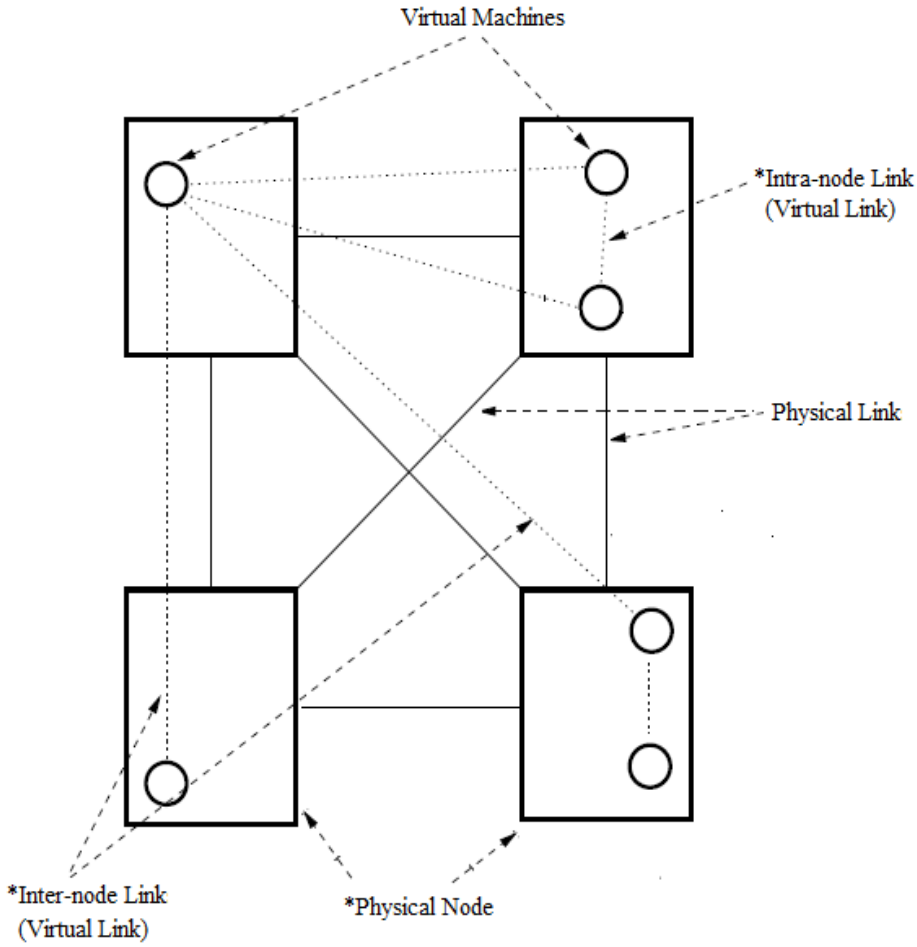

Figure 1: Testbed with 4 physical nodes and a sample virtual network.

2.3 The Effect of Virtual Networks on Testbed

From the above specification we know that virtual networks, when allocated, impose the following load on the physical machines in the testbed.

1. *Memory load.* This is a hard constraint which has to be met. Unless, otherwise a virtual machine will not execute.

2. *Computation load.* This is a soft constraint. As the aggregate CPU load increases on a physical machine the actual number of CPU cycles allocated for each virtual machine decreases. We assume that the amount of network traffic generated by each virtual machine is proportional to amount of CPU cycles allocated to the virtual machine. So decrease in CPU cycles allocated decreases the amount of total traffic generated by a virtual machine.

3. *Communication load*. This is also a soft constraint. We assume that intra-node bandwidth is very high and hence assume that intra-node links do not face any performance bottlenecks due to physical network bandwidth bottlenecks. The physical network bandwidth is assumed to be shared fairly between inter-node links. Inter-node links face

performance bottlenecks when they compete with each other for physical network bandwidth. This in turn effects the CPU utilisation of the virtual machine.

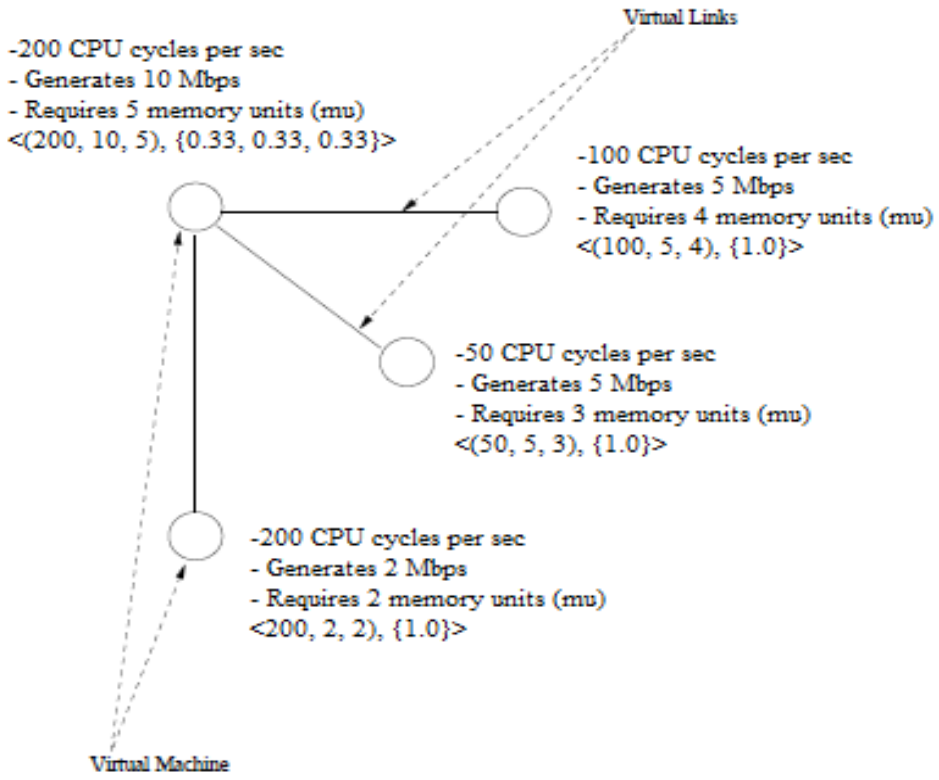

Figure 2: Specification of a virtual network.

So the testbed faces the classic network overlay problem of efficient allocation of virtual machines on physical resources. In simpler terms, when we overlay a virtual network onto physical machines we would like to balance and minimize the traffic induced by the overlay network on the physical network and at the same time balance load of virtual machines on the physical nodes. The objective of the overlay scheme is to maximize the efficiency of usage of computing resources and hosting of virtual networks.

Resources encompass network bandwidth of physical links, CPU and memory of physical nodes. If the overlay scheme is not fair then some of the physical nodes might be overloaded when compared to other machines. This will affect the performance of virtual networks whose virtual machines are hosted on the affected physical machines. Since V-NetLab is a distributed system, it's only fair to assume, that the overlay scheme should judiciously try to balance the load on the test-bed machines. At the same time, if a particular network is spread across too many physical nodes, the traffic induced by the overlay due to the communication between virtual machines might be high. This might lead to congestion on the physical links sooner than expected. Also care should be taken not to overload too many virtual links on to a single physical link. Thus the overlay scheme should also try to reduce and balance the traffic induced on the physical network as much as possible.

Figure 3 below shows a virtual network consisting of 6 virtual machines overlaid on a 2 node testbed. Also the specification of each virtual machine is given. Figure 4 shows best possible allocation virtual machines on the testbed. We assume that the memory capacity of each physical node is 600 memory units. In Figure 4 the memory constraint is met on both the physical nodes and the total CPU requirements are balanced. Also the traffic induced on the physical network has been minimized. Hence in the latter case the overlay network is expected to have maximum possible performance.

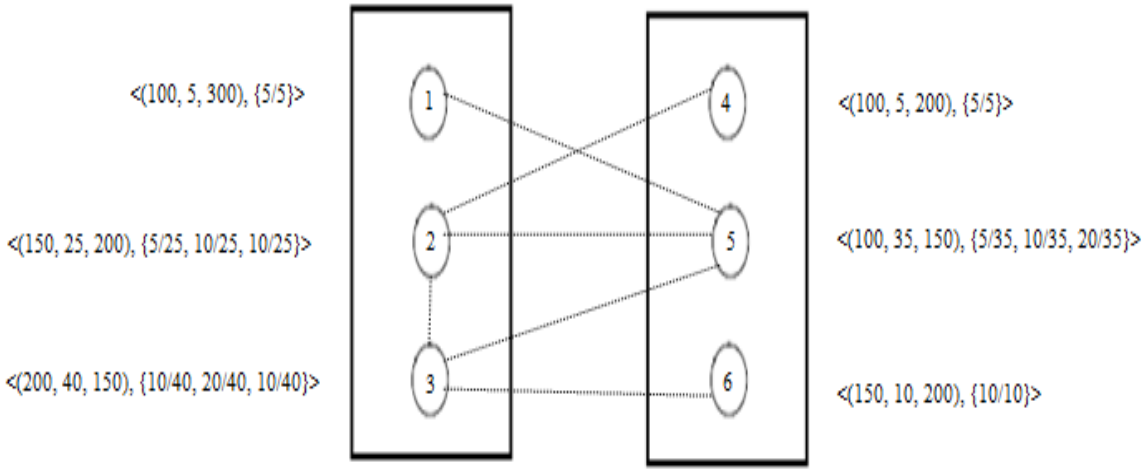

Figure 3: A sample 2 node testbed with a virtual network overlaid

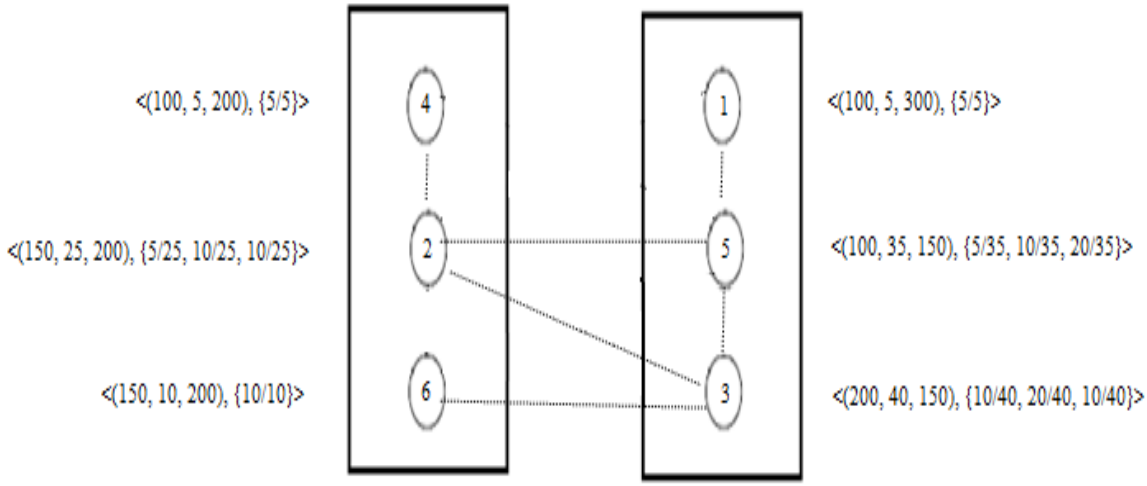

Figure 4: A better allocation of virtual network on the tested given in Figure 3

3 Allocation Problem

The network overlay or allocation problem can be precisely stated as follows:

- We have 'n' virtual machines where each virtual machine 'V_i' requires a fraction of physical machine CPU cycles 'C_i' per second and a fraction of physical memory 'M_i' to execute.

- We have ‘p’ physical machines each associated with a number of CPU cycles ‘C’ per second and a bound on physical memory available ‘M’. Every physical machine is connected to every other physical machine with a physical link of bandwidth ‘B’.
- We also have ‘m’ communicational links between various virtual machines. Each communicational link is associated with a link cost ‘B_i’ proportional to its bandwidth.
- We need to finally maximize the completion rate of all the networks which is calculated from completion rate of all the virtual machines allocated on physical machines without violating the memory constraints.
- Completion rate of a virtual machine allocated on a physical machine is calculated by taking into account the total computation and communication load imposed on the physical machine with respect to other physical machines.

We don’t assume any specific traffic characteristics among virtual machines and deal only with link costs. We assume that the completion rate of all the networks can be maximized by balancing the computation and communication load across all the physical machines.

The above problem can be stated in terms of Graph Partitioning as follows:

Given A graph G = (V, E) with (possibly unitary) weights on the edges and vertices and a parameter p,

Find A partitioning of the vertices of G into p sets in such a way that the sum of the vertex weights and the sum of the edge weights crossing between sets in each set are as equal as possible and total sum of edge weights crossing between the sets is minimized.

We basically deal with two cases of the above problem – the online allocation problem and the optimal allocation problem.

4 Optimal Allocation

In the optimal allocation problem, we know before hand all network topologies we would like to overlay. To solve this problem we employ simulated annealing method. We also use a multilevel algorithm for graph partitioning as a pre-pass to our simulated annealing method. The multilevel algorithm has following weaknesses:

1) It does not consider memory constraints of the physical machines.
2) It does not balance the inter-nodal links across the physical machines.

But it provides a good initial solution to the simulated annealing algorithm thereby minimizing its total running time.

4.1 Multilevel Algorithm

The multilevel algorithm provides solution to a graph partitioning problem similar to the one stated above. It deals with the following problem:

Given A graph G = (V, E) with (possibly unitary) weights on the edges and vertices and a parameter p,

Find A partitioning of the vertices of G into p sets in such a way that the sums of the vertex weights in each set are as equal as possible and the sum of the weights of edges crossing between sets is minimized.

The multilevel algorithm has three phases. First, a sequence of increasingly coarse approximations to a graph is constructed. Second, the smallest graph in the sequence is partitioned. A spectral method for this problem is used, but in principle any partitioning algorithm could be used. Third, the coarse partition is projected back through the sequence of graphs, periodically improving it with a local refinement algorithm. For this local improvement phase a variant of a popular algorithm originally devised by Kernighan and Lin and improved by Fiduccia and Mattheyses is used. Figure 5 depicts the multilevel algorithm stated above.

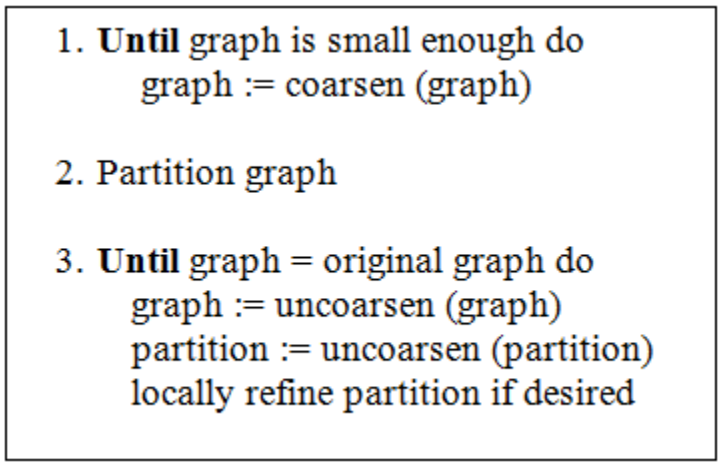

Figure 5: Multilevel Algorithm

The entire algorithm can be implemented to execute in time proportional to the size of the original graph. The costs of constructing coarse graphs and of the local improvement algorithms are both proportional to the number of edges in the graph. On the coarsest graph, moving vertices between partitions corresponds to moving a large collection of vertices on the original graph. Thus as we traverse the sequence of graphs, we make increasingly more refined corrections to the partition.

4.1.1 Constructing a coarse graph

The fundamental step in our coarsening scheme is the edge contraction operation. In this step, a maximal matching in the graph is identified and two vertices joined by an edge in the maximal matching are merged. The new vertex retains edges to the union of the neighbors of the merged vertices. The weight of the new vertex is set equal to the sum of the weights of its constituent vertices. Edge weights are left unchanged unless both merged vertices are adjacent to the same neighbor. In this case the new edge that represents the two original edges is given a weight equal to the sum of the weights of the two edges it replaces. Figure 6 depicts this step.

Coarsening is advantageous because the number of possible partitions grows exponentially with the number of vertices in the graph, so a good partition of the coarse graph is much easier to find than a good partition of the original one. The price paid for this reduction in complexity is that only a small number of the possible fine graph partitions are represented and are therefore examinable on the coarse graph. However, by working on different levels, the local

refinement scheme improves the partition on multiple scales. Although the partition of the coarse graph may not directly induce a very good partition of the original graph, the repeated application of the local refinement largely resolves this problem.

```
1. Find a maximal matching in the graph

2. For Each matching edge (i, j) do
       contract edge to form a new vertex v
       vertex weight (v) := weight (i) + weight (j)
       If i and j are both adjacent to a vertex k Then
           edge weight (v, k) := weight (i, k) + weight (j, k)
```

Figure 6: Constructing a coarse graph

4.1.2 Partitioning the coarsest graph

Multilevel algorithm uses a spectral partitioner for the coarse graph which requires one, two or three eigenvectors of the Laplacian matrix of a graph to partition into two, four or eight sets respectively. This is followed by an invocation of the Kernighan_Lin refinement algorithm.

4.1.3 Uncoarsening the partition

The uncoarsening of a partition is trivial. Each vertex in a coarse graph is simply the union of one or two vertices from a larger graph. A vertex from the larger graph is simply assigned to the same set as its coarse graph counterpart. Consequently, the best partition for the coarse graph may not be optimal for its un-coarsened counterpart. Therefore the local refinement scheme described in the next section is applied to the uncoarsened partition.

4.1.4 Locally refining the partition

We use a local refinement scheme that is fast, effective and a generalization of the graph bisection algorithm originally due to Kernighan and Lin (KL). The algorithm is summarized in Figure 7.

The gain is simply the net reduction in the weight of cut edges that would result from switching a vertex to a different set. Rather than simply choosing whichever of these moves has the highest gain, the algorithm also considers the effect of each move on set balance. This algorithm has two types of termination criteria. The most obvious condition is when there are no possible moves from large to small sets. The second condition allows for early termination when further improvement seems unlikely. Since there are typically many different vertex moves that have the same gain value, the manner in which ties are broken can significantly affect the results. This issue is avoided by using randomization.

This algorithm is quite good at finding locally optimal answers, but unless it is initialized with a good global partition, the resulting local optimum can be far from the best possible.

However, since the multilevel nature of our algorithm allows the refinement technique to work on different scales, the local finesse of KL is all that is needed.

```
Until No better partition is discovered
   Best Partition := Current Partition
   Compute all initial gains
   Until Termination criteria reached
      Select vertex to move
      Perform move
      Update gains of all neighbors of moved vertex
      If Current Partition balanced and better than Best Partition Then
          Best Partition := Current Partition
   End Until
   Current Partition := Best Partition
End Until
```

Figure 7: Locally refining the partition

This multilevel method takes somewhat longer but it produces partitions better in quality compared to other methods. One shortcoming of the algorithm is that the construction of a sequence of graphs is memory intensive.

The output of the multilevel algorithm is passed to simulated annealing method as input.

4.2 Simulated Annealing

Simulated annealing is an enhanced version of local optimization or iterative improvement technique, in which an initial solution is repeatedly improved by making small local alterations until no such alteration yields a better solution. Simulated annealing randomizes this procedure in a way that allows for occasional uphill moves (changes that worsen the solution), in an attempt to reduce the probability of becoming stuck in a poor but locally optimal solution and thus provide significantly better results.

The general algorithm for simulated annealing is given in Figure 8. Performance of annealing method depends on the initial solution picked, annealing schedule chosen, correctness of the cost metric and other, more problem-specific parameters. Simulated annealing avoids entrapment in poor local optima by allowing an occasional uphill move, done under the influence of a random number generator and a control parameter called the temperature T. The simulated annealing approach involves a pair of nested loops and two additional parameters, a cooling ratio r, $0 < r < 1$, and an integer temperature length L. Note that $e^{-A/T}$ will be a number in the interval (0, 1) when A and T are positive, and rightfully can be interpreted as a probability that depends on A and T. The probability that an uphill move of size A will be accepted diminishes as the temperature declines, and, for a fixed temperature T, small uphill moves have higher probabilities of acceptance than large ones. The greater the annealing schedule, the better are the solutions obtained. The term frozen refers to a state in which no further improvement in cost(S) seems likely.

1. Get an initial solution S.
2. Get an initial temperature $T > 0$.
3. While not yet *frozen* do the following.
   - 3.1 Perform the following loop L times.
     - 3.1.1 Pick a random neighbor S' of S.
     - 3.1.2 Let $\Delta = cost\ (S') - cost\ (S)$.
     - 3.1.3 If $\Delta \leq 0$ (downhill move), Set $S = S'$.
     - 3.1.4 If $\Delta > 0$ (uphill move), Set $S = S'$ with probability $e^{-\Delta/T}$.
   - 3.2 Set $T = rT$ (reduce temperature).
4. Return S.

Figure 8: Simulated Annealing basic algorithm

4.2.1 Solution

This simulated annealing algorithm produces an allocation in which the following conditions are satisfied.

1) CPU loads are balanced.
2) Inter-node links are minimized.
3) Inter-node links are balanced.
4) Memory constraints are met.

4.2.2 Initial Solution

Initial solution is generated from the pre-pass phase by using the multilevel algorithm.

4.2.3 Cost Metric

$$\text{Cost} = \text{Max}\ (\ \text{Cost}_i\ )\ \text{for i = 1 to P}$$

$$\text{Cost}_i = \left[\frac{\sum_{j=1}^{n_i} C_{ij}}{C} + \frac{\sum_{j}^{m_i} B_{ij}}{B} \right]$$

$$\text{Total Cost} = \sum_{i=1}^{P} (\ \text{Cost}_i\ )$$

Figure 9: Cost Metric

Figure 9 shows the cost metric used in optimal and online algorithms.

- P is the number of physical machines on the testbed.
- C is the number of CPU cycles executed by a physical machine per second.
- C_{ij} is the number of CPU cycles per second required by virtual machine j allocated on physical machine i.
- n_i is the number of virtual machines on physical machine i.
- B is the network bandwidth of physical machine i.
- B_{ij} is the bandwidth of an inter-nodal link j on physical machine i.
- m_i is the number of inter-node links from physical machine i.

The above cost metric is built based on the following assumptions:

1. Computing power of a physical machine is equally shared among all the virtual machines allocated on the physical machine.
2. The link bandwidth of a physical machine is equally shared among all the inter-nodal links.
3. Bandwidth requirements of intra-nodal links are fully met.

Following are the observations made from the above cost metric:

1. The cost metric does not favour balancing the load on a physical machine across other physical machines if the CPU load on the machine is much less compared to CPU capacity of the physical machine.
2. The cost metric favours balancing the load on a physical machine across other physical machines as the CPU load increases on the physical machine.
3. The cost metric favours balancing the load on a physical machine across other physical machines if the aggregate inter-nodal virtual link load on the machine is much less compared to physical link bandwidth.
4. The cost metric does not favour balancing the load on a physical machine across other physical machines as the aggregate inter-nodal virtual link load increases on the physical machine.

We look for an allocation which minimizes the cost metric. We also look for allocations which have lesser total cost in case the max cost is same.

4.2.4 Neighbor Selection

Initially we fix the physical machines where there is a memory violation by moving or exchanging the virtual machines currently allocated on them with those of physical machines where memory constraint has not been violated. This is done in a reliable way, since memory constraint is a hard constraint. We also construct a set of neighbors for each physical machine based on the connectivity of virtual machines and mark all the physical machines on which memory constraint has been violated. The following is the algorithm to select a new neighbor solution in each iteration of simulated annealing algorithm.

1. Randomly pick one marked physical machine. Do not pick the one picked in the previous step.
2. Swap virtual machines with the neighbors, choose the swaps which reduce the max cost or reduce the total cost and find a neighbor solution.
3. If VMs allocated on the selected machine change mark all its neighbors.
4. Unmark the selected machine.

5. If no swaps reduce the cost or no tainted machine found then
6. Randomly pick a machine with CPU load greater than the mean CPU load on the testbed and balance the CPU load with other machines. This can be done by swapping virtual machines which reduce the max cost or reduce the total cost. In case the max cost or total cost is not reduced then make swaps so as to balance CPU load by increasing the cost minimally.
7. Randomly pick a machine with inter-node link load greater than the mean inter-node link load on the testbed and balance with other physical machines starting with the one with least inter-node link load. This can be done by swapping virtual machines which reduce the max cost or reduce the total cost. In case the max cost or total cost is not reduced then make swaps so as to balance inter-node link load by increasing the cost minimally.
8. The new costs obtained in step 4 and 5 are compared and whichever produces a better allocation it is selected as the new neighbor solution.
9. Whenever VMs allocated on a machine change mark that machine.
10. In all the previous steps care is taken not to violate the memory constraint.

Simulated annealing has its own limitations. First is the question of running time. Many researchers have observed that simulated annealing needs large amounts of running time to perform well and this may push it out of the range of feasible approaches for some applications. Second is the question of adaptability. There are many problems for which local optimization is an especially poor heuristic, and even if one is prepared to devote large amounts of running time to simulated annealing, it is not clear that the improvement will be enough to yield good results.

The pre-pass algorithm initializes the simulated annealing with a good solution which reduces the length of the annealing schedule.

5 Online Allocation

The virtual networks are expected to arrive at irregular intervals of time on the testbed. Every time a new virtual network arrives we find an allocation which best optimizes the completion rate of the allocated networks on the testbed But doing a best allocation every time need not necessarily lead to an optimal allocation after n allocations as we don't intend to migrate networks across various physical machines. This is what we call by "Online algorithm". In the Online case we don't know the networks we might have to allocate in the future.

5.1 Requirement

1) Has to be fast.
2) Has to provide the solution to the problem.
3) Should serve the online nature.

5.2 Steps

In the following algorithm we make use of a similar cost metric decribed in optimal allocation algorithm.

1) The load imposed by previously allocated virtual machines on each physical machine is calculated and correspondingly the target size for each physical machine is set using a goal array. Then the multilevel algorithm is run only on the incoming new network using these target sizes for each physical machine.

2) We then fix the physical machines where there is a memory violation by moving or exchanging the new virtual machines currently allocated on them with those of physical machines where memory constraint has not been violated. This is done in a reliable way, since memory constraint is a hard constraint.
3) We then for each physical machine swap its new virtual machines with its neighbors and choose the swaps which reduce the max cost or reduce the total cost to produce a new optimized solution.
4) We don't balance CPU loads here since it is already done by the multilevel algorithm and not disturbed by the online algorithm. The cost metric takes care of maintaining the balance of CPU loads in the online algorithm. This is not the case with simulated annealing algorithm since it accepts certain uphill changes and may disturb the balance of CPU loads. This will increase the execution time of the online algorithm.
5) We then balance the inter-node links by considering physical machines in decreasing order of their inter-node link loads with physical machines in increasing order of their inter-node link loads.
6) In steps from 4 to 6, care is taken such that the memory constraint is not violated.

6 Measurement Algorithm

To measure the performance of the allocations produced by both the online and optimal algorithms, a measurement algorithm has been developed. In either case it measures the minimum completion rate of all the virtual machines currently allocated on the testbed. As stated earlier, we assume that the aggregate network traffic generated by a virtual machine in unit time is proportional to the number of CPU cycles executed per unit time.

Before stating the algorithm let us define two parameters – $alpha_i$ and $beta_i$.

$alpha_i$ = (Total CPU load / CPU capacity on physical machine i)
$beta_i$ = (Total inter-node link load / physical network bandwidth on physical machine i)

Following is the algorithm.

1. *Applying alpha (0, 1]:*

```
For each physical machine do
        Calculate alpha
        If alpha <= 1 continue
        For each virtual machine allocated on this physical machine do
                Divide CPU load of virtual machine by alpha
                For each virtual link incident from this virtual machine do
                        Divide virtual link bandwidth by alpha
```

2. *Applying beta (0, 1]:*

```
For each physical machine do
        Calculate alpha
        if alpha <= 1, alpha := 1
        Calculate beta and divide it by alpha
```

```
if beta <= 1 continue
Divide each inter-node link bandwidth by beta
```

3. *Choose minimum of the bandwidths:*

```
For each virtual machine x do
  For each virtual link incident from vm x to vm y
    xtoy_bw := bandwidth of virtual link incident from vm x to vm y
    ytox_bw := bandwidth of virtual link incident from vm y to vm x
    if ytox_bw < xtoy_bw
      xtoy_bw := ytox_bw
```

4. *Adjust CPU utilization.* For each virtual machine adjust the CPU load so that the aggregate network traffic it generates is proportional to the CPU load.

5. *Find minimum completion rate.* Compare completion rates of all the virtual machines and find the minimum.

We don't adjust CPU utilization and virtual link bandwidths of virtual machines if the physical machine is underutilized after step 4 as it is difficult to predict the dependencies between the virtual machines. Hence this algorithm provides completion rate of the testbed in worst case scenario.

7 Simulation

The simulator is simply an implementation of the aforementioned model. It comprises of the following modules:

1. *Network topology generation.* We use synthetic network topologies generated using IGen, a tool for network topology generation.
2. *Multilevel algorithm.* We use a modified version of existing open source code which implements the multilevel algorithm.
3. *Optimal module.* This module implements the optimal algorithm discussed above.
4. *Online module.* This module implements the online algorithm discussed above.
5. *Parser.* This module reads in all the input files like the network configuration file and files which contain information about the virtual networks already allocated on the testbed.

Figure 10 gives a block diagram of the simulator. We run the simulator for various network loads to assess the performance of online algorithm versus the performance of optimal allocation algorithm. Each time a new network arrives first the optimal algorithm is executed and the obtained results are logged. Next the online algorithm is executed and again the obtained results are logged. In both the cases, current testbed load file is read which contains the information about previously allocated networks. Only in the online case the multilevel algorithm reads in an extra current testbed allocation file which contains allocation information about the previous allocated networks. The results obtained in both the cases are compared. Since for each newly incoming network both the algorithms are executed only the online algorithm writes back to the current testbed load and current testbed allocation files.

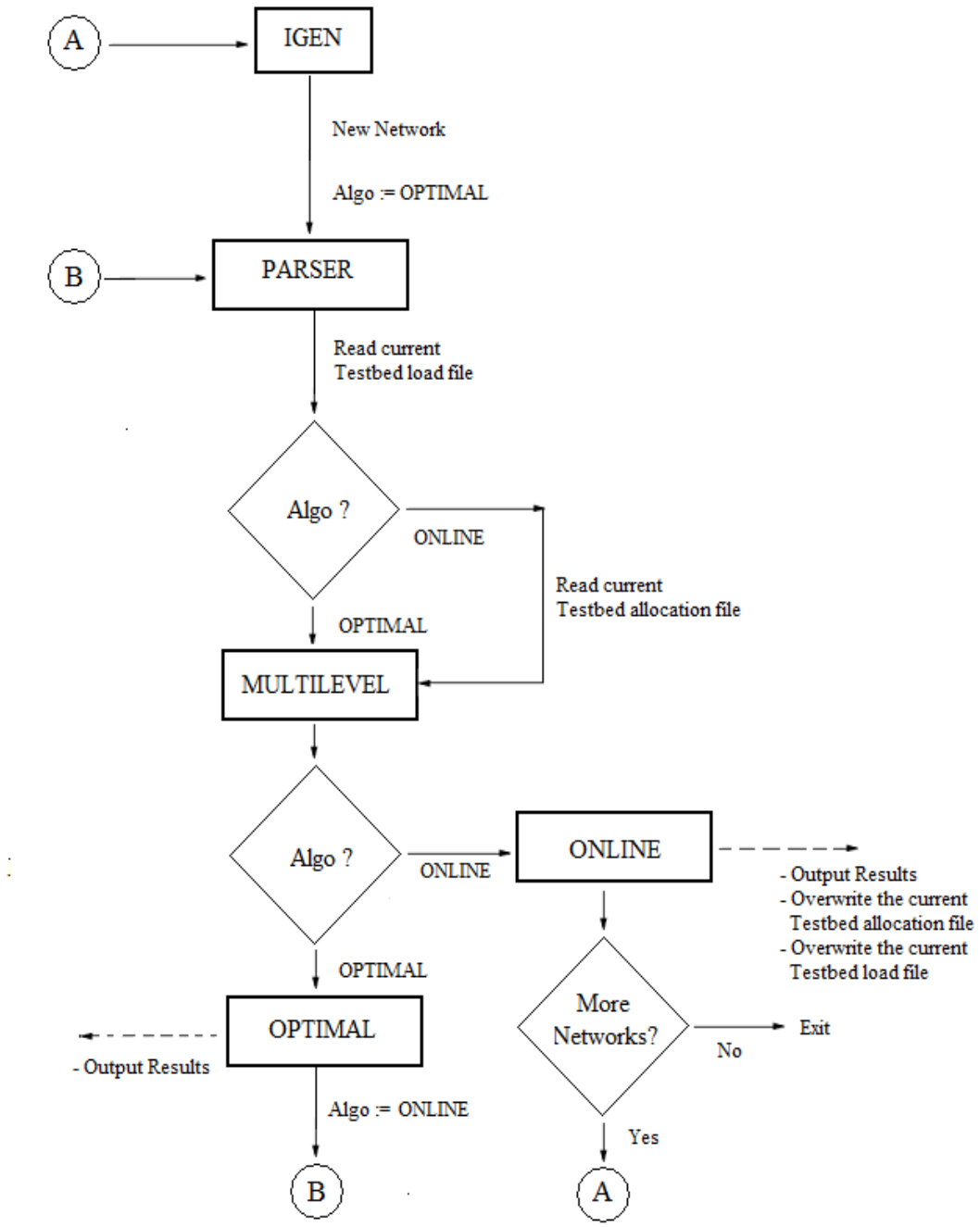

Figure 10: Block diagram of simulator

7.1 Network Topology Generation

To synthetically generate network topologies we use IGen, a tool for network topology generation. There are various other tools for network topology generation like Inet, GT-ITM, Brite etc. which produce topologies that respect graph properties seen in the real Internet. GT-

ITM for instance allows to build router level topologies with a backbone/access hierarchy. Nodes are placed randomly on a map and connected using a probabilistic model such as Waxman [47]. The problem of this approach is that topologies are generated in order to mimic pure graph properties of real networks. They fail to capture the optimization process that is also at the basis of the real network topologies.

The objectives of network design may be summarized in minimizing the latency, dimensioning the links so that the traffic can be carried without congestion, adding redundancy so that rerouting is possible in case of link or router failure and, finally, the network must be designed at the minimum cost. None of these objectives are currently explicitly found in degree-based generators such as BRITE [29] or GT-ITM [7].

Unlike GT-ITM and BRITE, IGen does not rely on probabilistic methods to generate network topologies. IGen implements various network design heuristics such as K-Medoids, MENTOR, MENTour, Delaunay triangulation and Two Trees for the purpose of building network topologies to cater to the various objectives of network design summarized above.

Basically, IGen methodology follows a bottom-up approach. The following are the steps to construct a network topology.

1. *Clustering* – The nodes are randomly placed in a plane and grouped into clusters called PoPs (Points of Presence) of the network using K-Medoids method. This method reduces the latency between access routers and backbone routers.

2. *PoP Structure* – The PoP structure is made robust to failures using redundant links between access routers and backbone routers.

3. *Backbone topology generation*- MENTOR, MENTour, Two Trees and Delaunay triangulation methods are used to produce a cost-effective topology with redundancy.

4. *IGP weights and capacities are assigned* - In order to assign realistic link capacities, IGen computes All-Pairs Shortest Paths and simulates the forwarding of traffic demand between all pairs of nodes to compute the utilization of each link. Based on the computed load of each link, the tool selects the suitable capacities.

We analyzed the CPU capacities of various real time backbone and access routers and assigned CPU capacities to each node which are considered as vertex costs in the graph partitioning algorithm. Since we need to optimize inter-nodal links with greater bandwidths, we consider bandwidths of each link as link cost in the graph partitioning algorithm. We generate network topologies using IGen for random number of nodes.

7.2 Input files and Output files

The following is the format of the input network file generated by IGen tool.
% comments here….
<number of vms> <number of virtual links> <option parameter>
<vm id> <vm CPU cycles per second> <virtual links list>
<vm id> <vm CPU cycles per second> <virtual links list>
.
.

<virtual links list> is specified as below:
<vm id> <virtual link bandwidth> <vm id> <virtual link bandwidth> ……

The vm id's are generated in increasing order. The option parameter specifies to the parser whether the network file contains CPU requirement for each virtual machine and whether each virtual link has certain bandwidth associated with it or not. In terms of graph partitioning, the option parameter specifies whether the graph file contains vertex and edges weights or not. We calculate memory requirement of each virtual machine randomly between a range of reasonable values.

Following is an example network file generated with 6 virtual machines and 7 virtual links. In this network virtual machine 1 with 20 CPU cycles per second requirement is connected to virtual machines 2, 5 and 6 with link bandwidths 10, 10 and 20 respectively.

% Generated by IGen
6 7 111
1 20 2 10 5 10 6 20
2 40 1 10 3 10
3 5 2 10 4 10
4 25 3 10 5 20
5 45 1 10 4 20 6 10
6 45 1 20 5 10

The format of the current testbed load file is the same as the above input network file. It contains all the networks already allocated on the testbed. Each time a new network is allocated, it is appended to this file.

Following is the format of current testbed allocation file.

<number of physical machines> <number of virtual machines>
<physical machine id> <vm id> <vm id> <vm id> …..
<physical machine id> <vm id> <vm id> <vm id> …..
.
.

This file contains information about all the virtual machines allocated on a particular physical machine. There is an entry for each physical machine in this file against which a list of virtual machines allocated on that physical machine is provided. The results file generated by each algorithm contains the completion rate of all the networks allocated on the testbed, during various stages of the algorithm along with the debug output if any.

7.2 Multilevel algorithm user input parameters

We used a modified version of existing open source code which implements the multilevel algorithm. The following are some of the user input parameters set in configuration files to get the desired functionality.

1. *PHY_MACH_NUM*. This parameter determines the number of set sizes into which the input graph should be partitioned by the multilevel algorithm. It determines the number of physical machines available on the testbed.

2. *ARCHITECHTURE*. This parameter determines the architecture of the physical machines available on testbed. It accepts either hypercube or mesh architectures. We set the architecture to two-dimensional mesh. We also set the sizes of each dimension (X_EXTENT and Y_EXTENT). The total number of sets (X_EXTENT * Y_EXTENT) should be equal to *PHY_MACH_NUM* parameter.

3. *PARTITION_DIM.* This parameter determines the dimensionality of the partitioning scheme. Most graph partitioning codes rely on recursive bisection. That is the graph is partitioned into two pieces, each of these pieces is partitioned into two more, etc. but the multilevel algorithm also partitions graph into 4 or eight sets at each stage of recursion which seem to provide better solutions in some graph partitioning algorithms. Partitioning the graph into 4 or 8 sets at each stage provides solution faster than the bisection case. After partitioning the total number of cuts between the partitioned sets is minimized. Since we want to minimize the total number of cuts as much as possible we prefer partitioning into 2 sets over 4 or eight sets. Experiments also support that in our graph partitioning problem the bisection case provides solutions compared to others.

4. *OPTIMAL_ONLINE*. This parameter controls which algorithms have to be run when a new network arrives. A value of 1 will run only the optimal algorithm. A value of 2 will run only the online algorithm. A value of 3 will run both optimal and online algorithm.

5. *GLOBAL_PART_METHOD.* This parameter determines which partitioning method should be run. Apart from multilevel algorithm the open source code implements other similar graph partitioning algorithms like spectral, inertial etc. This parameter is set to value 1 so that multilevel algorithm is executed.

6. *Network file.* The simulator generates input graph files with names "graph1.rig", "graph2.rig" etc. 'n_{th}' incoming network file has "graphn.rig" name. We used this convention for simplicity.

7. *Assignment output file*. If *OUTPUT_ASSIGN* parameter is set to TRUE then the allocation is outputted to "Online_assign.out" file and "Optimal_assign.out" files in case of online and optimal algorithms respectively.

8. *Results output file*. The partitioning results like total set size, maximum set size, minimum set size, total edge cuts, maximum edge cuts per set, minimum edge cuts per set and timing results are outputted to "Online_results.out" and "Optimal_results.out" files in case of online and optimal algorithms respectively.

7.3 Simulation Results

8 Assumptions and Limitations

8.1 Assumptions

The following are the assumptions made in this implementation:

1. Intra-node links. There is no degradation in performance of a virtual machine due to intra-node virtual links. The intra-node bandwidth is infinity.
2. Fair CPU scheduling. A physical machine fairly schedules the virtual machines based on their CPU requirement.
3. Fair bandwidth utilization. The network bandwidth of a physical machine is uniformly shared among all the inter-node virtual links.
4. There is no congestion, packet loss or any other specific traffic characteristic among virtual machines which degrades the bandwidth of virtual links.
5. Completion rate is maximized by balancing the computation and communication load across all the physical machines when load on physical machines is greater than the capacity of the physical machines.
6. The degradation of processor utilization with increase in memory load is ignored. Since we are considering memory requirement as a hard constraint, it is reasonable to have this assumption in place.
7. All the physical machines in the testbed are identical.

8.2 Limitations

The following are the limitations of this implementation:

1. The optimal or online algorithm tries to move or exchange only a single virtual machine to reduce the cost metric. Ideally we may need to move or exchange a group of machines to reduce the cost metric.
2. A better cost metric can be developed.
3. The multilevel algorithm produces good initial solutions only if the memory requirement is proportional to the CPU requirement of the virtual machine. To fix this, the multilevel algorithm implementation possibly needs to be changed so that it takes in a tuple <cpu load, memory load> instead of just the cpu load. Also the cost metric of multilevel algorithm may be modified to balance the inter-nodal virtual link load.

9 Future Work

10 Bibliography

1. 'A Multilevel Algorithm for Partitioning Graphs'. http://www.sandia.gov/~bahendr/abstracts/multilevel.html

2. 'Large-scale Virtualization in the Emulab Network Testbed'. http://www.cs.utah.edu/flux/papers/virt-usenix08-base.html.

3. 'Optimization by Simulated Annealing: An Experimental Evaluation; Part 1, Graph Partitioning' by Johnson D. S. , Aragon C. R. , Mcgeoch L. A. ;and Schevon C.

4. B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. Bell Sys. Tech. J., 49(2):291–308, 1970.

5. 'Towards More Representative Internet Topologies' – (igen-tech-report) by Bruno Quoitin.