VADAPT: Algorithms and Details

Ananth I. Sundararaj Ashish Gupta Peter A. Dinda
{ais,ashish,pdinda}@cs.northwestern.edu
Department of Computer Science, Northwestern University

1 Virtuoso

Virtuoso is a system for virtual machine grid computing that for a user very closely emulates the existing process of buying, configuring, and using an Intel-based computer or collection of computers from a web site. The mechanisms in Virtuoso that are salient to this description are VNET, our virtual networking system, and VTTIF, our application topology and traffic inference framework. VADAPT, is our adaptation control system.

2 Adaptation and VADAPT

Virtuoso uses VTTIF to determine the communication behavior of the application running in a collection of VMs and can leverage the plethora of existing work on network monitoring ([1] is good taxonomy) to determine the behavior of the underlying resources. The VNET component of Virtuoso provides the mechanisms needed to adapt the application to the network. Beyond this, what is needed is

- the measure of application performance, and
- the algorithms to control the adaptation mechanisms in response to the application and network behaviors

Here the measure is the throughput of the application.

The adaptation control algorithms are implemented in the VADAPT component of Virtuoso. VADAPT uses greedy heuristic algorithms to quickly answer this question when application information is available, and VM migration and topology/forwarding rule changes are the adaptation mechanisms.

2.1 Topology adaptation

VADAPT uses a greedy heuristic algorithm to adapt the VNET overlay topology to the communication behavior of the application. VTTIF infers the application communication topology giving a traffic intensity matrix that is represented as an adjacency list where each row describes communication between two VMs. The topology adaptation algorithm is as follows:

1. Generate a new list which represents the traffic intensity between VNET daemons that is implied by the VTTIF list and the current mapping of VMs to hosts.
2. Order this list by decreasing traffic intensity.
3. Establish the links in order until \( c \) links have been established.

The cost constraint \( c \) is supplied by the user or system administrator. The cost constraint can also be specified as a percentage of the total intensity reflected in the inferred traffic matrix, or as an absolute limit on bandwidth.

Following is the topology adaptation algorithm, described above in English, in formal notation. The call \( \text{AdaptTopology}(A, B, c) \), where \( A \) is the VM adjacency list, \( B \) is the VM to host mapping and \( c \) is the cost constraint, produces an ordered set \( S \) of
fasts-path links that need to be added.

**Procedure** AdaptTopology\((A, B, c)\)

1. \( k \leftarrow 0 \)
2. **while** \( count \neq \text{SizeOf}(A) \) **do** {loop invariant: \( count \leq \text{SizeOf}(A) \)}
3. \[
\begin{align*}
C[k, 0] &\leftarrow B(A[k, 0]) \\
C[k, 1] &\leftarrow B(A[k, 1]) \\
C[k, 2] &\leftarrow A[k, 2] \\
k &\leftarrow k + 1
\end{align*}
\]
4. **end while**
5. Quicksort\((C)\)
6. \( count \leftarrow 0 \)
7. \( i \leftarrow 0 \)
8. \( k \leftarrow 0 \)
9. **while** \( count \neq c \) and \( count \leq \text{SizeOf}(A) \) **do** {loop invariant: \( count \leq c \)}
10. \[
\begin{align*}
S[k] &\leftarrow \text{LINK } A[i, 0] \ A[i, 1] \\
k &\leftarrow k + 1 \\
i &\leftarrow i + i \\
count &\leftarrow count + 1
\end{align*}
\]
11. **end while**
12. Return \( S \)

### 2.2 Migration

VADAPT uses a greedy heuristic algorithm to map virtual machines onto physical hosts. As above, VADAPT uses the application communication behavior as captured by VTTIF and expressed as an adjacency list as its input. In addition, we also use throughput estimates between each pair of VNET daemons arranged in decreasing order. The algorithm is as follows:

1. Generate a new list which represents the traffic intensity between VNET daemons that is implied by the VTTIF list and the current mapping of VMs to hosts.
2. Order the VM adjacency list by decreasing traffic intensity.
3. Order the VNET daemon adjacency list by decreasing throughput.
4. Make a first pass over the VM adjacency list to locate every non-overlapping pair of communicating VMs and map them greedily to the first pair of VNET daemons in the VNET daemon adjacency list which currently have no VMs mapped to them. At the end of the first pass, there is no pair of VMs on the list for which neither VM has been mapped.
5. Make a second pass over the VM adjacency list, locating, in order, all VMs that have not been mapped onto a physical host. These are the “stragglers”.
6. For each of these straggler VMs, in VM adjacency list order, map the VM to a VNET daemon such that the throughput estimate between the VM and its already mapped counterpart is maximum.
7. Compute the differences between the current mapping and the new mapping and issue migration instructions to achieve the new mapping.

Following is the migration adaptation algorithm, described above in English, in formal notation. The call \( \text{MapVM}(A, B, G) \), \( A \) is the VM adjacency list, \( B \) is the VNET daemon adjacency list and \( G \) is the VM to host mapping, produces a mapping from VMs to physical hosts.
Procedure MapVM(A, B, G)

1: \( k \leftarrow 0 \)
2: while \( \text{count} \neq \text{SizeOf}(A) \) do \{loop invariant: \( \text{count} \leq \text{SizeOf}(A) \}\)
3: \[
E[k, 0] \leftarrow G(A[k, 0]) \\
E[k, 1] \leftarrow G(A[k, 1]) \\
E[k, 2] \leftarrow A[k, 2] \\
k \leftarrow k + 1
\]
4: end while
5: Quicksort(E)
6: Quicksort(B)
7: size \leftarrow \text{SizeOf}(E)
8: \( i \leftarrow 0 \)
9: \( l \leftarrow 0 \)
10: Comment: \( C \) contains the VM nodes that have already been mapped, currently empty
11: Comment: \( D \) contains the host nodes that have already been mapped to, currently empty
12: Comment: \( T \) contains the new mapping from VMs to host nodes, currently empty
13: while \( i \neq \text{size} \) do \{loop invariant: \( i \leq \text{size} \}\)
14: \[
\begin{align*}
m &\leftarrow 0 \\
\text{if } E[i, 0]\notin C &\text{ and } E[i, 1]\notin C \text{ then} \\
C[m] &\leftarrow E[i, 0] \\
C[m + 1] &\leftarrow E[i, 1] \\
m &\leftarrow m + 2 \\
l &\leftarrow 0 \\
\text{while } j \neq \text{SizeOf}(B) &\text{ do } \{\text{loop invariant: } j \leq \text{SizeOf}(B)\}\}
\end{align*}
\]
15: \[
\begin{align*}
n &\leftarrow 0 \\
\text{if } B[j, 0]\notin D &\text{ and } B[j, 1]\notin D \text{ then} \\
D[n] &\leftarrow B[i, 0] \\
D[n + 1] &\leftarrow B[i, 1] \\
n &\leftarrow n + 1 \\
l[0] &\leftarrow E[i, 0] \\
l[1] &\leftarrow B[j, 0] \\
l &\leftarrow l + 1 \\
T[l, 0] &\leftarrow E[i, 1] \\
T[l, 1] &\leftarrow B[j, 1] \\
l &\leftarrow l + 1
\end{align*}
\]
16: end if
17: end while
18: \( i \leftarrow i + i \)
19: end while
20: \( i \leftarrow 0 \)
21: while \( i \neq \text{size} \) do \{loop invariant: \( i \leq \text{size} \}\)
22: \[
\begin{align*}
\text{if } E[i, 0]\notin C &\text{ then} \\
C[m] &\leftarrow E[i, 0] \\
m &\leftarrow m + 1
\end{align*}
\]
\begin{verbatim}
44:  j ← 0
45:  found ← 0
46:  while j ≠ SizeOf(T) and found == 0 do {loop invariant: j ≤ SizeOf(T)}
47:      {
48:          if E[i, 0] == T[j, 0] then
49:              temp ← T[j, 1]
50:              found ← 1
51:          end if
52:          if E[i, 0] == T[j, 1] then
53:              temp ← T[j, 0]
54:              found ← 1
55:          end if
56:      }
57:  end while
58:  T[l, 0] ← E[i, 0]
59:  T[l, 1] ← temp
60:  l ← l + 1
61:  if E[i, 1] ≠ C then
62:      C[m] ← E[i, 1]
63:      m ← m + 1
64:      j ← 0
65:  found ← 0
66:  while j ≠ SizeOf(T) and found == 0 do {loop invariant: j ≤ SizeOf(T)}
67:      {
68:          if E[i, 1] == T[j, 0] then
69:              temp ← T[j, 1]
70:              found ← 1
71:          end if
72:          if E[i, 1] == T[j, 1] then
73:              temp ← T[j, 0]
74:              found ← 1
75:          end if
76:      }
77:  end while
78:  T[l, 0] ← E[i, 1]
79:  T[l, 1] ← temp
80:  l ← l + 1
81:  count ← count + 1
82: end while
83: Return T
\end{verbatim}

2.3 Forwarding rules

Once VADAPT determines the overlay topology, we compute the forwarding rules using an all pairs shortest paths algorithm with each edge weight corresponding to the total load on the edge from paths we have determined. This spreads traffic out to improve network performance.
2.4 Combining algorithms

When we combine our algorithms, we first run the migration algorithm to map the VMs to VNET daemons. Next, we determine the overlay topology based on that mapping. Finally, we compute the forwarding rules.

References