A Technique for improving the scheduling of network communicating processes in MOSIX

Rengakrishnan Subramanian
Masters Report, Final Defense

Guidance by Prof. Dan Andresen
Agenda

- MOSIX
  - Network communicating processes
  - Breaking it down
  - Timing
  - Implementation
  - Test
  - Results
  - Conclusion
MOSIX - purpose

- Software tool for cluster computing
- Multiple servers work as if single to achieve high performance
- Automatic work distribution
- Load balancing
- Adaptive management (processes v/s resources)
- Create a process (or more processes)
- Distribute (and redistribute)
- Algorithms respond to variation
- Works on Linux x86 platforms
- Kernel patch
- System Admin tools
Agenda

- MOSIX
- **Network communicating processes**
- Breaking it down
- Timing
- Implementation
- Test
- Results
- Conclusion
Network Processes

- Preemptive process migration
- Interaction with environment after migration
- User context, System / UHN context
- Remote migrates, deputy stays at UHN
- Deputy has kernel resources (sockets too!)
Network Processes - origin

Node a
Process A & B

Cluster of nodes

Node b

Node a
Process A

Cluster of nodes

Node b
Process B
Network Process - migrated
Network Process - migrated
Agenda

- MOSIX
- Network communicating processes
- Breaking it down
- Timing
- Implementation
- Test
- Results
- Conclusion
Breaking down – Microscopic
Agenda

- MOSIX
- Network communicating processes
- Breaking it down
- **Timing**
  - Implementation
  - Test
  - Results
- Conclusion
Timing - picture

- User Space
- Kernel / Socket Layer
- TCP/IP
- Firewall
- Lower Layers

Timing:
- \( T_1 \): 11.94 µsec
- \( T_2 \): 8.59 µsec
- \( T_3 \): 24.22 µsec
- \( T_4 \): 2.9 µsec
- \( T_5 \)
- \( T_6 \)
- \( T_7 \)
- \( T_8 \)
- \( T_9 \)
The amount of time that can be saved if the packets are redirected at the firewall layer = 

(Time taken at the TCP layer, 24.22 µsec) 

+ (Time taken at the socket layer, 2.9 µsec) 

+ (Time taken by MOSIX to decide on the fate of the packet, M µsec).
Recalculated Time =
\[ 2 \times \{ 
\text{(Time taken at the TCP layer, 24.22 \, \mu\text{sec}) } + 
\text{(Time taken at the socket layer, 2.9 \, \mu\text{sec})} 
\} + 
\text{(Time taken by MOSIX to decide on the fate of the packet, M \, \mu\text{sec})}. \]
Timing - Inference

- Yes, time can be saved
- Yes, time can be saved at the firewall layer
- The packet can be identified
- The packet can be redirected
- Because, the firewall can do NAT
Agenda

- MOSIX
- Network communicating processes
- Breaking it down
- Timing
- **Implementation**
- Test
- Results
- Conclusion
Implementation - IPTables

- Built-in firewall tool in Linux kernel
- Successor of IPChains and IPFwadm
- MOSIX works on Linux x86
- Timing available for IPTables
Implementation - NAT

- **PRE-Routing DNAT**
- **Routing Decision**
- **POST-Routing SNAT**
- **Local Process**
Implementation - picture

Node b
Firewall

Node a
Client

Node c
Server
Implementation - rules

- # iptables -t nat -A PREROUTING -s $CLIENT -d $FIREWALL_SYSTEM -p tcp -dport $SERVER_PORT -i eth0 -j DNAT --to-destination $NEW_DESTINATION

- # iptables -t nat -A POSTROUTING -s $CLIENT -p tcp --dport $SERVER_PORT -o eth0 -j SNAT --to-source $FIREWALL_SYSTEM

- # iptables -t nat -A PREROUTING -s $SERVER -d $FIREWALL_SYSTEM -p tcp --sport $SERVER_PORT -i eth0 -j DNAT --to-destination $CLIENT
- MOSIX
- Network communicating processes
- Breaking it down
- Timing
- Implementation
- Test
- Results
- Conclusion
Testing – what?

- MOSIX communication through redirection
- IPTables communication through redirection
- Direct communication
Testing - Environment

- Pentium P4 CPU
- 1.6 GHz Processor speed
- Intel Ether express Network card
- 100 Mbps LAN
- Two Red Hat 7.2 Linux boxes with Kernel 2.4.19
- One Debian Linux box with Kernel 2.4.18
- All nodes were connected on to the same LAN switch
Testing – how?

- Server-Client communicating pair
- Parameterized
- Buffer size
- Amount of data → time
- Number of such communicating pairs
- Port numbers
- Prints out time taken for data transfer
Node b
Server is created here.
Server waits on a port number.

Node a
Client will reside here
Client is not yet created.

Node c
Testing – MOSIX 2

Node b
Server is migrated manually using MOSIX admin tools to node c
All processes think server is still in Node b

Node a
Client is created.
Contacts server in Node b.
Is unaware that server is in Node c

Node c
Server is now here.
But, it goes to Node b for system calls. Node b is its UHN
Testing - IPTables

Node b
IPTables rules are written here.
Forwards packets from Node a to Node c and vice-versa

Node a
Client is created here
Client contacts Node b requesting for service

Node c
Server is created here.
It will get request from Node b (which is in reality from Node a).
It will reply back to Node b
Node a
Client is created here
Client contacts Node c
requesting for service

Node b

Node c
Server is created here.
It will get request from Node a
It will reply back to Node a
Agenda

- MOSIX
- Network communicating processes
- Breaking it down
- Timing
- Implementation
- Test
- Results
- Conclusion
### Results – Execution time

<table>
<thead>
<tr>
<th>No end-to-end connections</th>
<th>LATENCY (sec)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MOSIX</td>
<td>IPTABLES</td>
<td>NORMAL</td>
</tr>
<tr>
<td>3</td>
<td>203.73</td>
<td>109.79</td>
<td>103.51</td>
</tr>
<tr>
<td>6</td>
<td>275.56</td>
<td>219.58</td>
<td>212.61</td>
</tr>
<tr>
<td>9</td>
<td>390.28</td>
<td>328.89</td>
<td>316.60</td>
</tr>
<tr>
<td>12</td>
<td>513.64</td>
<td>437.77</td>
<td>424.55</td>
</tr>
<tr>
<td>15</td>
<td>640.92</td>
<td>552.14</td>
<td>529.11</td>
</tr>
<tr>
<td>25</td>
<td>1063.21</td>
<td>913.89</td>
<td>882.72</td>
</tr>
<tr>
<td>50</td>
<td>2130.70</td>
<td>1840.99</td>
<td>1746.64</td>
</tr>
</tbody>
</table>
## Results - Bandwidth

<table>
<thead>
<tr>
<th>No end-to-end connections</th>
<th>BANDWIDTH (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MOSIX</td>
</tr>
<tr>
<td>3</td>
<td>15.71</td>
</tr>
<tr>
<td>6</td>
<td>11.61</td>
</tr>
<tr>
<td>9</td>
<td>8.19</td>
</tr>
<tr>
<td>12</td>
<td>6.23</td>
</tr>
<tr>
<td>15</td>
<td>4.99</td>
</tr>
<tr>
<td>25</td>
<td>3.01</td>
</tr>
<tr>
<td>50</td>
<td>1.50</td>
</tr>
</tbody>
</table>
## Results – CPU Utilization

<table>
<thead>
<tr>
<th>No end-to-end connections</th>
<th>%cpu utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MOSIX</td>
</tr>
<tr>
<td>3</td>
<td>58.8</td>
</tr>
<tr>
<td>6</td>
<td>85.7</td>
</tr>
<tr>
<td>9</td>
<td>85</td>
</tr>
<tr>
<td>12</td>
<td>87.3</td>
</tr>
<tr>
<td>15</td>
<td>87.5</td>
</tr>
<tr>
<td>25</td>
<td>90</td>
</tr>
<tr>
<td>50</td>
<td>90</td>
</tr>
</tbody>
</table>
Results – Load Average

<table>
<thead>
<tr>
<th>No end-to-end connections</th>
<th>Load average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MOSIX</td>
</tr>
<tr>
<td>3</td>
<td>0.52</td>
</tr>
<tr>
<td>6</td>
<td>1.34</td>
</tr>
<tr>
<td>9</td>
<td>1.73</td>
</tr>
<tr>
<td>12</td>
<td>1.83</td>
</tr>
<tr>
<td>15</td>
<td>1.8</td>
</tr>
<tr>
<td>25</td>
<td>3.55</td>
</tr>
<tr>
<td>50</td>
<td>4.42</td>
</tr>
</tbody>
</table>
Results – Execution Graph

Execution Time Comparison

Time (secs)

No of Connections

mosix  iptables  direct
Results – Bandwidth Graph

Bandwidth Comparison

Bandwidth (Mbps)

No of connections

- mosix
- iptables
- direct
Results – CPU Graph

%CPU Utilization Comparison

%CPU Utilization

No of connections

0 10 20 30 40 50 60

0 20 40 60 80 100

mosix  iptables
Results – Load Graph

Load Average Comparison

No of Connections

Load Average

- mosix
- iptables
Agenda

- MOSIX
- Network communicating processes
- Breaking it down
- Timing
- Implementation
- Test
- Results
- Conclusion
MOSIX takes 33% more time on execution than direct communication.

IPTables takes only 4% more.

MOSIX takes 28% more time on execution than IPTables.
Bandwidth of MOSIX is 20% less than IPTables

MOSIX, on an average, takes 212% more CPU utilization

MOSIX takes at least 138 times more load average than IPTables
Yes, better performance can be achieved

- Better Execution time
- Better Bandwidth utilization
- Better load average
- Better CPU utilization