Pradeep Sundararajan, Abhishek Gupta, Mathew Ryden, Abhishek Chandra, Jon Weissman

Department of CS&E
University of Minnesota
Outline

• Conventional cloud
• Limitations and opportunities
• Nebula project
The “Standard” Cloud

Data in

Computation

Results out

“No limits”
- Storage
- Computing
Current Cloud Model

• Largely centralized
• Pay-as-you-go
• Strong guarantees
• 3rd party
Appealing Features

• Scale/consolidation
  – elasticity, lower TCO

• Strong locality
  – data and computing => great for analytics

• Novel sharing platform
  – data/state and applications => gaming, Web 2.0
Fraying at the Edges

• Privacy
  – don’t want everything going to the cloud but some things

• Social/community networks
  – limited sharing

• Locality
  – largely centralized cloud => bottlenecks
    • to users ...
    • to/from data sources ... (think: Big Data)
Big Data Trend

• Big data is distributed
  – earth science: weather data, seismic data
  – life science: GenBank, NCI BLAST, PubMed
  – health science: GoogleEarth + CDC pandemic data
  – web 2.0: user multimedia blogs
  – “everyone is a sensor”
Privacy/Locality Trend

• Privacy
  – restrict/filter data (think: patient records)

• Locality
  – mobile users: latency sensitive application access
  – criticality: “deliver go-signal to my insulin pump”
Need New Features

• Process data in-situ or close by
  – save time and money
  – privacy-aware

• Organize platform based different notions of “closeness”
  – network distance
  – trusted nodes
  – social groups
  – communities of interest
Idea

• Make the cloud more “distributed”
  – “move” it closer to data
  – “move” it closer to end-users
  – “move” it closer to other clouds
Example: Dispersed-Data-Intensive Services

- Data is geographically distributed
  - Costly, inefficient to move to central location
Example: Blog Analysis
Nebula: A New Cloud Model

• Stretch the cloud
  – exploit the rich collection of edge computers
  – volunteers (P2P, @home), commercial (CDNs)
Nebula

• Decentralized, less-managed cloud
  – dispersed storage/compute resources
  – low user cost
Example: Blog Analysis
Blog Results

- Amazon emulator
- Nebula testbed

<table>
<thead>
<tr>
<th># Blogs</th>
<th>Time taken (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>20000</td>
</tr>
<tr>
<td>80</td>
<td>40000</td>
</tr>
<tr>
<td>120</td>
<td>60000</td>
</tr>
<tr>
<td>160</td>
<td>80000</td>
</tr>
<tr>
<td>240</td>
<td>120000</td>
</tr>
<tr>
<td>320</td>
<td>140000</td>
</tr>
</tbody>
</table>
Failure Resistant

<table>
<thead>
<tr>
<th>Total Number of Blogs</th>
<th>Time Taken (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>480</td>
</tr>
<tr>
<td>560</td>
<td>640</td>
</tr>
<tr>
<td>720</td>
<td>800</td>
</tr>
</tbody>
</table>

- CCE - 0 Failures
- Nebula - 0 Failures
- Nebula - 1 Failure
- Nebula - 2 Failure
- Nebula - 3 Failure
Another Example: Latency-Sensitive

- Mobile service
How is Nebula different from @home?

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Nebula</th>
<th>@home</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collective performance</td>
<td>High</td>
<td>None</td>
</tr>
<tr>
<td>Locality/Context-awareness</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Statefulness</td>
<td>High/medium</td>
<td>Low</td>
</tr>
</tbody>
</table>
Common Service Characteristics

• Elastic resource consumption
  – scale up/down based on demand

• Geographical data/user distribution
  – execution dependent on location of data/user

• Weak performance/robustness requirements
  – some failures may be tolerable
Inside Nebula

- Nebula central
- Chrome
- Dashboard
- Datastore
Nebula Central

- Manager
- Volunteers check-in
- Tracks global state of other services
- Distributes code and nebula software
- Run at UMn
- Central point of trust
DataStore

• Data service that runs on subset of nodes
• Provides basic store/retrieval
• Policy-based management for a specific DS
  – capacity, latency, fault tolerance, durability

<table>
<thead>
<tr>
<th>Operation</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_create_group</td>
<td>datastore_id, source, destination, scoring_function, number</td>
<td>Create data node group</td>
</tr>
<tr>
<td>client_put</td>
<td>source, datastore_id, file_id, number</td>
<td>Put a file in a data group</td>
</tr>
<tr>
<td>client_get</td>
<td>datastore_id, file_id</td>
<td>Fetch a file from a data group</td>
</tr>
</tbody>
</table>
SecureNode

- Nebula nodes run a Chrome Browser
  - secure sandbox (NaCL) native client inside
  - all native code executes inside it

Figure 11: Performance overhead of NativeClient
Network Dashboard

- **Software tool** `netstat.cs.umn.edu`
- Runs on all nebula nodes
- Provides point-to-point latency, jitter, bandwidth
- Used by DataStore service, NodeGroup service (future)
Dashboard Output

(a) TCP bandwidth (KBps) (Higher is better)

(b) UDP bandwidth (KBps) (Higher is better)

(c) UDP delay (ms) (Lower is better)

(d) UDP jitter (ms) (Lower is better)
Summary

• Nebula: new cloud architecture
  – Preserves cloud behavior: APIs, elasticity, transparency
  – Stronger notion of external locality
  – Weaker notion of internal locality

• Future work
  – End-to-end system operational
  – Connect to the commercial cloud
    “use the edge opportunistically”
Thank you! Questions?