Exploring Data Parallelism and Locality in Wide Area Networks

Yunhong Gu
Univ. of Illinois at Chicago

Robert Grossman
Univ. of Illinois at Chicago and Open Data Group
Overview

- Cloud Computing and Sector/Sphere Stack
- Sphere Computing Paradigm
- System Design and Implementation
- Applications and Performance Evaluation
Cloud Computing & Sector/Sphere

- **Hardware**
  - Inexpensive computer cluster
  - High speed wide area networks

- **Software**
  - Storage cloud: Distributed, big data set
  - Compute cloud: simplified distributed data processing
Sector: Distributed Storage System

Security Server
- User account
- Data protection
- System Security

Master
- Storage System Mgmt.
- Processing Scheduling
- Service provider

Client
- System access tools
- App. Programming
- Interfaces

SSL

slaves

Data

UDT
- Encryption optional

Storage and
Processing
Sphere: Simply Data Processing

- Data is processed at where it resides (locality)
- Same user defined functions (UDF) can be applied on all elements (records, blocks, or files)
- Processing output can be written to Sector files, on the same node or other nodes
- Generalize Map/Reduce
Sphere: Simplify Data Processing

for each file F in (SDSS dataset)
  for each image I in F
    findBrownDwarf(I, ...);

SphereStream sdss;
  sdss.init("sdss files");
SphereProcess myproc;
  myproc->run(sdss,"findBrownDwarf");
  myproc->read(result);

findBrownDwarf(char* image, int isize, char* result, int rsize);
Sphere: Data Movement

- Slave -> Slave
  - Local
- Slave -> Slaves (Shuffle)
- Slave -> Client
Load Balance & Fault Tolerance

- The number of data segments is much more than the number of SPEs. When an SPE completes a data segment, a new segment will be assigned to the SPE.
- If one SPE fails, the data segment assigned to it will be re-assigned to another SPE and be processed again.
Example: Sorting a TeraByte

- Data is split into small files, scattered on all slaves.
- Stage 1: On each slave, an SPE scans local files, sends each record to a bucket file on a remote node according to the key, so that all buckets are sorted.
- Stage 2: On each destination node, an SPE sorts all data inside each bucket.
TeraSort

Binary Record 100 bytes

<table>
<thead>
<tr>
<th>10-byte</th>
<th>90-byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>Value</td>
</tr>
</tbody>
</table>

Stage 1:
Hash based on the first 10 bits

Stage 2:
Sort each bucket on local node

10-bit
0-1023

Bucket-0
Bucket-1
Bucket-1023
Bucket-1023
Implementation

- Sector/Sphere is written C++ and released as open source software
  - [http://sector.sf.net](http://sector.sf.net)
  - Current release version 1.5

- TeraSort Example Application
  - 50 lines of C++ code, in addition to local sorting code
  - Prepare input files, output destinations, and call user defined functions (hash and sort)
Open Cloud Testbed

- 4 Racks in Baltimore (JHU), Chicago (StarLight and UIC), and San Diego (Calit2)
- 10Gb/s inter-site connection on CiscoWave
- 1Gb/s inter-rack connection
- Two dual-core AMD CPU, 12GB ram, 1TB single disk
Performance Results: TeraSort

Run time: seconds

<table>
<thead>
<tr>
<th></th>
<th>Data Size</th>
<th>Sphere</th>
<th>Hadoop (3 replicas)</th>
<th>Hadoop (1 replica)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UIC</td>
<td>300GB</td>
<td>1265</td>
<td>2889</td>
<td>2252</td>
</tr>
<tr>
<td>UIC + StarLight</td>
<td>600GB</td>
<td>1361</td>
<td>2896</td>
<td>2617</td>
</tr>
<tr>
<td>UIC + StarLight + Calit2</td>
<td>900GB</td>
<td>1430</td>
<td>4341</td>
<td>3069</td>
</tr>
<tr>
<td>UIC + StarLight + Calit2 + JHU</td>
<td>1.2TB</td>
<td>1526</td>
<td>6675</td>
<td>3702</td>
</tr>
</tbody>
</table>
Performance Results: TeraSort

- Sorting 1.2TB on 120 nodes
- Hash vs. Local Sort: 60% : 40%

**Hash**
- Per rack: 680GB in/out; Per node: 10GB in/out
- CPU: 50% MEM: 900MB

**Local Sort**
- No network IO
- CPU: xx MEM: XX

**Hadoop:** CPU 150% MEM 2GB
For More Information

- Sector/Sphere code & docs: [http://sector.sf.net](http://sector.sf.net)

- Open Cloud Consortium: [http://www.opencloudconsortium.org](http://www.opencloudconsortium.org)

- NCDM: [http://www.ncdm.uic.edu](http://www.ncdm.uic.edu)

- @SC08: Booth 321
CreditStone

**Stage 1:**
Process each record and hash into buckets according to merchant ID

**Stage 2:**
Compute fraudulent rate for each merchant

<table>
<thead>
<tr>
<th>Trans ID</th>
<th>Time</th>
<th>Merchant ID</th>
<th>Fraud</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>01491200300</td>
<td>2007-09-27</td>
<td>2451330</td>
<td>0</td>
<td>66.49</td>
</tr>
</tbody>
</table>
**Performance Results: CreditStone**

<table>
<thead>
<tr>
<th>Racks</th>
<th>JHU</th>
<th>JHU, SL</th>
<th>JHU, SL, Calit2</th>
<th>JHU, SL, Calit2, UIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Nodes</td>
<td>30</td>
<td>59</td>
<td>89</td>
<td>117</td>
</tr>
<tr>
<td>Size of Dataset (GB)</td>
<td>840</td>
<td>1652</td>
<td>2492</td>
<td>3276</td>
</tr>
<tr>
<td>Size of Dataset (rows)</td>
<td>15B</td>
<td>29.5B</td>
<td>44.5B</td>
<td>58.5B</td>
</tr>
<tr>
<td>Hadoop (min)</td>
<td>179</td>
<td>180</td>
<td>191</td>
<td>189</td>
</tr>
<tr>
<td>Sector with Index (min)</td>
<td>46</td>
<td>47</td>
<td>64</td>
<td>71</td>
</tr>
<tr>
<td>Sector w/o Index (min)</td>
<td>36</td>
<td>37</td>
<td>53</td>
<td>55</td>
</tr>
</tbody>
</table>

* Courtesy of Jonathan Seidman of Open Data Group.
Inverted Index

Stage 1:
Process each HTML file and hash (word, file_id) pair to buckets

HTML page_1

<table>
<thead>
<tr>
<th>word_x</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>word_y</td>
<td>1</td>
</tr>
<tr>
<td>word_z</td>
<td>1</td>
</tr>
</tbody>
</table>

1st letter

Stage 2:
Sort each bucket on local node, merge same word

<table>
<thead>
<tr>
<th>word_z</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>word_z</td>
<td>5</td>
</tr>
<tr>
<td>word_z</td>
<td>10</td>
</tr>
</tbody>
</table>

Bucket-A

Bucket-B

Bucket-Z

Bucket-A

Bucket-B

Bucket-Z

word_z | 1, 5, 10